FACULTAD DE INFORMATICA
UNLP

TESINA DE LICENCIATURA

TITULO: Análisis de aplicabilidad de la herramienta Rational Team Concert para la definición de procesos de software

AUTORES: de Hormaechea, Sebastián Omar - Silvera, Juan Francisco

DIRECTOR: Dra. Claudia Pons

CODIRECTOR:

CARRERA: Licenciatura en Informática

Resumen

La utilización correcta de procesos para el desarrollo de software, puede determinar críticamente la calidad de un producto final. Estos métodos suelen ser efectivos y eficientes, y generalmente mejoran la productividad y la calidad de un producto de software. Sin embargo, una mala definición, elección y/o aplicación del proceso resulta totalmente contraproducente. Históricamente, primero se frecuentaba el uso de "Metodologías Tradicionales", pero ante las necesidades actuales de eficiencia y rapidez en el desarrollo de software, es que surgen las "Metodologías Ágiles".
IBM presenta el software Rational Team Concert como una herramienta innovadora que permite el manejo de ciclo de vida de aplicaciones ágiles (Utilizando SCRUM, OpenUp, etc.), disponiendo de un ambiente colaborativo entre los miembros de un equipo, proporciona un ámbito de desarrollo transparente a los usuarios, entre otras ventajas. Al mismo tiempo, se asegura que es óptimo para equipos pequeños y medianos.
En base a estas aseveraciones, y al tratarse de una herramienta reciente sobre la cual aun no existen reportes de experiencias concretas ni estudios de aplicabilidad, se justifica un análisis profundo de un software capaz de integrar miembros de un equipo con roles asignados, agenda de trabajo, eficiencia, rapidez, un ambiente colaborativo, entre otras cualidades necesarias, en una actualidad donde se exige calidad y el tiempo apremia, para poder competir en el mercado. Rational Team Concert, según IBM, posee estas ventajas.
El objetivo de esta tesis fue realizar, a modo de "Feedback", un análisis de la herramienta RTC mediante el desarrollo de un caso concreto basado en SCRUM y OpenUp.

Líneas de Investigación

- Ingeniería de software
- Metodologías tradicionales
- Metodologías ágiles
- Procesos Scrum y OpenUp
- Herramientas para el desarrollo de software
- Plataforma Jazz
- Producto Rational Team Concert

Trabajos Realizados

Luego de un estudio profundo (Procesos Scrum y OpenUp, Plataforma Jazz y el software RTC), se analizó el software Rational Team Concert mediante el desarrollo de un caso de estudio concreto basado en Serum y OpenUp. Se compara la definición formal de Serum y OpenUp, con la forma en la que los utiliza RTC. Se advierten falencias y funcionalidades que agrega RTC.
También se compara RTC con otras herramientas alternativas, desde puntos de vista objetivos y subjetivos. Se intenta determinar cuando es conveniente o no migrar a RTC.
Se hace un análisis crítico entre la herramienta RTC y las características formales que debe tener un buen software y se apoya o discute la información propiciada por IBM

Conclusiones

Como aporte de este trabajo se concluyó que RTC es una poderosa aplicación que provee buena performance de acceso y comunicación con el servidor, una gran flexibilidad en general, una fuerte administración de proyectos, procesos y usuarios, una amplia variedad de informes, una comunicación que favorece una estrecha colaboración entre miembros de un equipo, un registro de completo de eventos, amplia variedad de herramientas y funcionalidades útiles, posibilidad de administración desde una IDE o desde una interfaz Web, entre otras virtudes.
Es una excelente y completa aplicación, óptima para grupos medianos de desarrollo.

Trabajos Futuros

El software RTC presenta otras propiedades que también merecen su atención, para obtener, como ampliación de este informe, un análisis completo de todas las características que restan estudiar de la herramienta. Se proponen:
- Análisis de la integración y soporte de RTC con ClearQuest, ClearCase y Subversion.
- Análisis de la integración de RTC con RQM y RRC
- Análisis de la versión cliente de Visual Estudio
- Análisis de los templates de los otros procesos incluidos en el software (ej. "Eclipse Way")
- Análisis de seguridad de RTC

Fecha de la presentación: Mayo 2010

Tesina de grado

Análisis de aplicabilidad de la herramienta Rational Team Concert para la definición de procesos de software.

Autores
de Hormaechea, Sebastián
Silvera, Juan Francisco

Director
Dra. Claudia Pons

Índice de contenidos

Índice de figuras

Agradecimientos

Llegado este momento, comprendo que se esta cerrando una de las etapas mas importantes de mi vida. Como toda etapa, con decepciones pero también con sus logros, siendo éste seguramente, el más grande de todos.

Todo logro es de uno, pero la mayoría de las veces uno cuenta con personas que lo acompañan y que implícita o explícitamente, lo ayudan a concretarlo. A esas personas quiero agradecer.

Primero quisiera agradecer a aquellos profesores de la facultad que, por su pasión, entusiasmo y sabiduría, me aportaron muchos conocimientos.

Quiero agradecer a Claudia por aceptar coordinar ésta tesis, y por los aportes que sugirió para el desarrollo y la finalización del trabajo. También expresar mi gratitud a Valeria, que nos ayudó mucho y rápidamente con inconvenientes que surgieron. Agradecer a Francisco por aceptar compartir este trabajo.

A dos compañeros y amigos, Kike y Fercho, que durante el transcurso de la carrera, siempre me aportaron su ayuda incondicional.

A todos mis amigos del alma, a los que no podría nombrar uno por uno por temor a olvidarme de alguno, por todos los buenos momentos, y por estar siempre en los malos, dándome fuerzas para seguir. Haciendo una excepción quiero agradecer a Licho y Arol por ayudarme, y ser tan buenos amigos, y especialmente al Pali por estar siempre presente.

Quiero reservarme un agradecimiento muy profundo y especial a Naty por haber sido mi motor, mi pilar y mi oxigeno en este ultimo período de mi carrera; por levantarme, por entusiasmarme, por darme fuerzas, por contenerme, por inspirarme, por alentarme y por

haberme hecho vivir momentos tan felices. Todo éste trabajo, es producto de lo que ella me generó y me sigue generando.

Y agradecer profundamente a mis viejos, por educarme, darme la posibilidad de estudiar, por haber soportado tantas trabas y demoras, y sin embargo, estar siempre presente de manera incondicional con todo el amor y cariño que siempre me dieron.

Sebastián

Quiero agradecer a Sebastián por compartir la tesis y gran parte de mi carrera.

A Claudia, coordinadora de nuestra tesina.

A mis familiares y amigos por estar siempre.

Francisco

Capítulo 1

Introducción

Este apartado es la introducción general al informe. Se definen los objetivos de la tesina, la motivación del estudio de la herramienta y los resultados esperados a partir de la experiencia de utilizar el software.

1.1. Objetivo

IBM recientemente lanzó la herramienta de desarrollo de software Rational Team Concert (RTC), un software sobre el cual aun no existen reportes de experiencias concretas ni estudios de aplicabilidad.

El objetivo es realizar, a modo de "Feedback", un análisis de la herramienta (la cual corre sobre la plataforma Jazz, también de IBM) mediante el desarrollo de un caso concreto basado en Scrum y OpenUp.

1.2. Motivación

Aplicar un proceso de desarrollo de software generalmente es importante (incluso indispensable). La utilización correcta de estos procesos puede determinar críticamente la calidad del producto final, incluso en proyectos de pequeña envergadura. Estos métodos suelen ser efectivos y eficientes, y generalmente mejoran la productividad y la calidad de un producto de software.

Un objetivo de décadas ha sido el encontrar procesos y metodologías standards, que sean sistemáticas, predecibles y repetibles. Es ante esta necesidad que, en un principio, surgen diferentes modelos de procesos tales como el "Modelo en Cascada" o secuencial, "Modelo Espiral", "Modelo Iterativo Incremental" y sus diversas variantes, entre los mas conocidos.

Pero con el correr del tiempo, las necesidades de eficiencia, y rapidez en el desarrollo de software comenzaron a ser preponderantes (aun hoy lo son). Ante este nuevo escenario es

que surgen los denominados procesos ágiles de desarrollo de software. Estos procesos intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en la gente y los resultados. Sin embargo, son criticados y tratados como "indisciplinados" por la falta de documentación técnica, falta de diseño y falta de análisis.

Cabe aclarar que algunas organizaciones utilizan procesos de desarrollo de software propios en lugar de los formales o ágiles, e incluso hay algunas que no se basan en ningún proceso.

Una mala definición del proceso o una mala elección y aplicación del mismo resulta totalmente contraproducente (por ejemplo, si se trata de un proyecto de pequeña envergadura, probablemente convenga utilizar una metodología ágil).

IBM presenta Rational Team Concert como una herramienta innovadora que permite el manejo de ciclo de vida de aplicaciones ágiles (mediante la utilización de Scrum, OpenUp o un proceso de equipo simple), dispone de un ambiente colaborativo entre los miembros de un equipo (que pueden estar en diferentes lugares físicos), proporciona un ámbito de desarrollo transparente a los usuarios, entre otras ventajas. Al mismo tiempo, IBM asegura que su software es óptimo para equipos pequeños y medianos de desarrollo, pero que también soporta a los grandes grupos de trabajo (en su versión Enterprise). También afirma que el software soporta a proyectos de pequeña, mediana o gran envergadura.

En base a estas aseveraciones, y al tratarse de una herramienta reciente sobre la cual aun no existen reportes de experiencias concretas ni estudios de aplicabilidad, se justifica un análisis profundo de un software capaz de integrar miembros de un equipo con roles asignados, agenda de trabajo, eficiencia, rapidez, un ambiente colaborativo, entre otras cualidades necesarias, en una actualidad donde se exige calidad y el tiempo apremia, para poder competir en el mercado.

Rational Team Concert, según IBM, posee estas ventajas.

1.3. Resultados esperados

El resultado inmediato de esta tesis será exponer las características, ventajas y desventajas del software Rational Team Concert a partir de un profundo análisis y corroborar si el mismo permite a organizaciones de software llevar a cabo proyectos de calidad de manera organizada y eficiente.

Capítulo 2

Estructura del trabajo

Se optó por dividir el informe en capítulos ya que esta distribución aporta una mayor organización de la información, pudiendo así, resaltar conceptos, análisis o conclusiones de diferentes ítems que hacen al trabajo de grado.

Se creyó conveniente introducir el informe con conceptos necesarios para describir a Rational Team Concert y a las tecnologías que utiliza, para luego, entonces si, explayar conclusiones y reflejar experiencias de la utilización de dicha herramienta.

Cabe aclarar, que en un primer momento se había propuesto como capitulo 2, el desarrollo de conceptos relacionados a la ingeniería de software y a las metodologías tradicionales. Si bien es cierto que todo aporte es fructífero, se cree que este material excedía mucho el núcleo temático de la tesis, haciéndola muy extensa. De todas maneras se añade este capítulo, junto a la bibliografía en la versión en formato digital.

A continuación se explica brevemente el contenido de los capítulos realizados:

2.1. Introducción capítulo 3

RTC soporta metodologías ágiles para desarrollar un software. En el capitulo 3 se explaya sobre dichas metodologías.

A modo de reseña histórica, primero se explica el escenario en el que surge la necesidad de definir a los procesos ágiles.

Luego se presentan definiciones formales y conceptos relacionados a dichas metodologías, indicando objetivos, características, y enumerando procesos existentes.

También se explica el Manifiesto Ágil, documento propulsor de estas nuevas metodologías.

Se comparan las metodologías tradicionales y las metodologías ágiles, indicando características genéricas, ventajas y desventajas de cada una. También se intentan definir ciertos parámetros para considerar cuando es conveniente utilizar uno u otro tipo de metodología.

Finalmente se abordan de manera profunda y detallada los procesos que soporta RTC, Scrum y OpenUp, los cuales serán utilizados para realizar el caso de estudio.

2.2. Introducción capítulo 4

RTC se presenta como una herramienta eficaz para el desarrollo de software. Pero obviamente no es la única. Este capitulo presenta algunas herramientas alternativas para ese fin, indicando especificaciones técnicas, arquitectura, características, ventajas y desventajas de cada una. Se explican algunos software como subversión, Git, Mercurial, Team Foundation Server (Versión de Microsoft, competidor natural de IBM), entre otros.

Este apartado tiene como objetivo presentar otras herramientas aptas para el desarrollo de software y describir sus especificaciones y características.

2.3. Introducción capítulo 5

Este capitulo comienza a abordar la temática principal de la tesina. En este apartado se describen las características, arquitectura y especificaciones técnicas de la plataforma Jazz, así también como los objetivos que IBM busca alcanzar con ésta, software y plugins soportados, entre otros.

Finalmente se aborda en particular el software RTC, se especifica detalladamente aspectos técnicos, características, objetivos, ventajas, entre otros, como fundamentos por los cuales IBM recomienda el uso y promoción de su software.

Cabe aclarar que este capitulo busca principalmente exponer la información propiciada por IBM acerca de RTC. A partir de la experiencia realizada, y según las conclusiones arribadas, a modo crítico se podría apoyar o refutar dicha información (total o parcialmente), explayar características, plantear interrogantes, discutir información, etc.

2.4. Introducción capítulo 6

En este capitulo se explaya sobre el desarrollo de un caso de estudio concreto utilizando la herramienta RTC. Se propone el desarrollo de un software de pequeña/mediana envergadura para analizar las características generales de la herramienta RTC (también características puntuales).

Se decidió desarrollar el caso de estudio, utilizando y analizando exhaustivamente el proceso Scrum, ya que éste, es uno de los procesos ágiles más utilizados en la actualidad. Finalizado esto, se realizaron pruebas en forma más genérica del proceso OpenUp. Cabe aclarar que la mayoría de las pruebas acerca de las funcionalidades de RTC, están en el apartado dedicado a Scrum (6.5. Caso de estudio basado en Scrum). No se creyó necesario iniciar un proyecto completo en OpenUp, ya que, si bien existen diferencias en la definición de los templates, y otras diferencias menores no existen diferencias en las funcionalidades que brinda RTC.

En este apartado se compartirá la experiencia de utilizar RTC, se exponen características del software, problemas, novedades, ventajas etc.

2.5. Introducción capítulo 7

Aquí se plasman las conclusiones finales arribadas. Se busca compartir tanto enfoques objetivos como subjetivos (discutibles).

La intención general del apartado es solapar las características de la herramienta y la experiencia del uso de ésta, con lo desarrollado en los capítulos anteriores

Se compara la definición formal de los procesos ágiles Scrum y OpenUp, con la forma en la que los utiliza RTC. Se advierten falencias, funciones o herramientas agregadas, etc.

Se advierte por que es tan difícil comparar ésta herramienta con las otras herramientas alternativas.

Se intentó determinar cuando es conveniente o no utilizar RTC.

Se buscó apoyar, refutar, discutir o plantear interrogantes o dudas entre la información propiciada por IBM (por ejemplo cuando trata sobre las ventajas del producto) y la experiencia concreta de utilizar el software.

Finalmente se hace un análisis crítico entre la herramienta RTC y las características formales que debe tener un buen software (por ejemplo, en cuanto a portabilidad, facilidad de uso, soporte, etc.).

A modo ampliatorio, se realizó una entrevista a personal del laboratorio de investigación LIFIA, que utiliza el proceso ágil Scrum, con el fin de refinar y obtener nuevas conclusiones.

2.6. Introducción capítulo 8

En este capitulo, se plantean posibles futuras investigaciones relacionadas con la tesina desarrollada.

2.7. Otras consideraciones

En el informe, se presentan diversos gráficos y tablas (referenciados ambos como "figuras").

Además se exponen recuadros informativos (con fondo grisáceo). Estos recuadros sirven como ampliación de la información relativa al apartado donde aparecen. Esta información se cree aclarativa o ampliatoria.

Capítulo 3

Procesos ágiles. OpenUp y Scrum

RTC soporta metodologías ágiles para desarrollar un software. En el capitulo 3 se explaya sobre dichas metodologías.

A modo de reseña histórica, primero se explica el escenario en el que surge la necesidad de definir a los procesos ágiles.

Luego se presentan definiciones formales y conceptos relacionados a dichas metodologías, indicando objetivos, características, y enumerando procesos existentes.

También se explica el Manifiesto Ágil, documento propulsor de estas nuevas metodologías.

Se comparan procesos tradicionales y procesos ágiles, indicando características genéricas, ventajas y desventajas de cada uno. También se intentan definir ciertos parámetros para considerar cuando es conveniente utilizar uno u otro tipo de metodología.

Finalmente se abordan de manera profunda y detallada los procesos que soporta RTC, Scrum y OpenUp, los cuales serán utilizados para realizar el caso de estudio.

3.1. El nuevo escenario

A partir de la década del 80, el ciclo de vida de los proyectos era el denominado en cascada; El proyecto se divide en fases, y éstas se ejecutan de forma secuencial: definición del producto, diseño, construcción de elementos, integración, pruebas...

Dos características de la construcción de nuevos productos en ese tiempo son:

- Ciclo de vida secuencial.
- División y especialización del trabajo.

Cada fase la realiza un departamento, personas o equipos diferentes, profesionalmente especializados en los conocimientos necesarios.

La gestión de proyectos desarrolla modelos de estructuras organizativas de tipo matricial, con diferentes variaciones, para facilitar la comunicación y coordinación entre equipos diferentes.

Al mismo tiempo, a la par de la consolidación del conocimiento de gestión de proyectos, se estaban desarrollando las teorías de producción basada en procesos, como mejor medio para garantizar la calidad, eficiencia y repetitividad.

División del trabajo, especialización y producción basada en procesos, fueron premisas que, como axiomáticas, asumió la gestión de proyectos, y por esta razón, los puntos clave de la ***gestión predictiva o clásica*** son:

- Estimar cuál va a ser el trabajo necesario, y a continuación gestionar la ejecución para que se cumplan la estimación inicial.
- El trabajo se desarrolla en fases.
- División del trabajo en equipos de especialistas.
- Desarrollo basado en procesos.

Hasta entonces, el desarrollo de nuevos productos se realizaba como una carrera de relevos, en la que un grupo de especialistas funcionales pasaban el relevo al siguiente, pese a que se comenzó a evidenciar que los desarrollos secuenciales puros suelen ser más teóricos que prácticos, y en realidad quienes los adoptan, generalmente producen ciclos "secuenciales con solapamiento", donde una fase no suele necesitar para empezar que esté completamente terminada la anterior.

Sin embargo, cuando la teoría de gestión de proyectos estaba alcanzando una cierta madurez, se comenzó a observar que algunas empresas, en mercados muy competitivos, relacionados con productos de vanguardia tecnológica, trabajaban ignorando esa teoría. En estas empresas o bien no recorrían fases a través de diferentes equipos especializados, o bien estas se solapaban notablemente. (Figura 3.1)

Figura 3.1. Diferentes modelos de gestión de proyectos

En los primeros años los productos tardaban mucho en quedar obsoletos, y las empresas los producían con variaciones mínimas a lo largo del tiempo. Sin embargo, dos variables modificaron el escenario del desarrollo de nuevos productos, a finales del siglo pasado.

- **Velocidad:** A fines de siglo la vida de los productos empieza a ser más breve, y una vez desarrollados apenas se mantienen unos meses el catálogo de novedades, y enseguida quedan fuera del mercado. Los ingresos de las empresas no dependen ya de los productos veteranos, sino de los últimos desarrollos. El principal factor estratégico de varias empresas es innovación continua.

- **Incertidumbre:** En un escenario rápido e inestable, encajan mal las estrategias de negocio predictivas, que diseñan un producto y trazan un plan de negocio. En los entornos que evolucionan con rapidez las empresas que invierten trabajo y análisis para trazar estrategias y planes, deben cambiarlos constantemente para poder sobrevivir. En los entornos rápidos los productos y estrategias que alcanzan el éxito no son los que se desarrollan sobre planes pre-concebidos, sino los que crecen en adaptación y replanteamiento constante guiados por la evolución del propio entorno. La incertidumbre es una consecuencia de la velocidad. En los sectores sometidos a una evolución muy rápida, no es posible, ni aconsejable, trazar un plan de negocio para un producto en el momento de su diseño.

Antes el proceso de desarrollo llevaba asociada un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del buen hacer de la ingeniería del software, asumiendo el riesgo que ello conlleva.

En este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

Las metodologías ágiles son sin duda uno de los temas recientes en ingeniería de software que están acaparando gran interés. Prueba de ello es que se están haciendo un espacio destacado en la mayoría de conferencias y Workshops celebrados en los últimos años. En la comunidad de la ingeniería del software, se está viviendo con intensidad un debate abierto entre los partidarios de las metodologías tradicionales (referidas peyorativamente como "metodologías pesadas") y aquellos que apoyan las ideas emanadas de las "Metodologías Ágiles". Por un lado, para muchos equipos de desarrollo el uso de metodologías tradicionales les resulta muy lejano a su forma de trabajo actual considerando las dificultades de su introducción e inversión asociada en formación y herramientas. Por otro, las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos industriales de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías.

Para triunfar en el nuevo escenario, lo importante no es tener garantías de que se va a cumplir un plan inicial.

Las variables claves para triunfar en el nuevo escenario son:
- Tomar retro-información del producto y del entorno de forma continua.
- Dar el mayor valor innovador al producto.
- En el menor tiempo posible.
- Para salir lo antes posible al mercado
- No hay producto "terminado" el producto está en continua evolución

Estos productos no necesitan modelos de gestión predictivos, sino adaptables.

En un *modelo predictivo* el objetivo es lograr lo planificado: en un modelo de negocio predictivo el objetivo es producir el producto diseñado, con los costes y las ventas previstas en el plan de negocio.

En un *modelo adaptativo* el objetivo es dar al producto el mayor valor posible de forma constante. Como el escenario es muy rápido e inestable no es realista pensar que se podrá mantener el plan de negocio trazado al empezar. Es más realista pensar que se podrá mantener el producto con el mayor valor posible.

El plan de negocio pasa a ser plan de valor. El objetivo es el valor, y el negocio la consecuencia lógica si se dispone de un producto que mantiene su valor de forma continua.

El modelo adaptativo no pide por la predictibilidad de que el plan inicial se ejecute en las fechas y con los costes previstos, sino poder disponer en el menor tiempo posible de valor

para el cliente, y mantener el producto o servicio en evolución continua para incrementarlo, o al menos mantenerlo.

> Hoy determinados productos permanecen en un continuo estado "beta". El entorno tecnológico es tan inestable, que las novedades se lanzan tras el menor tiempo de desarrollo posible, dejando que vayan evolucionando a través de versiones, en el propio mercado. Que sea éste quien diga de forma continua cómo deben modificarse los "requisitos".
>
> En estas circunstancias, las diferencias de liderazgo entre unas empresas y otras no radica tanto en la eficiencia y previsibilidad con la que se gestionan el lanzamiento de nuevos productos, sino en la capacidad de agilidad y cambio durante su construcción; y el principal valor para ocupar puestos de cabeza es la innovación.

Es en éste contexto que comienzan a surgir y definirse a los *procesos ágiles de desarrollo de software*.

3.2. Procesos ágiles de desarrollo de software

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo.

Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Una posible mejora es incluir en los procesos de desarrollo, más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto.

Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Esta es la filosofía de las *metodologías ágiles*, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales.

Se entiende como Desarrollo ágil de software a un paradigma de Desarrollo de Software basado en procesos ágiles. Los procesos ágiles de desarrollo de software, conocidos anteriormente como metodologías livianas, intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en la gente y los resultados.

Es un marco de trabajo conceptual de la ingeniería de software que promueve iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto. Existen muchos métodos de desarrollo ágil; la mayoría minimiza riesgos desarrollando software en cortos lapsos de tiempo. El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener una "versión beta" (sin errores) cuando esta concluya. Luego, al final de cada iteración, el equipo vuelve a evaluar las prioridades del proyecto.

Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación. Entre los participantes se debe incluir revisores, diseñadores de iteración, escritores de documentación y ayuda, y directores de proyecto.

Los métodos ágiles también enfatizan que el software funcional es la primera medida del progreso. Combinado con la preferencia por las comunicaciones cara a cara, generalmente

los métodos ágiles son criticados y tratados como "indisciplinados" por la falta de documentación técnica.

Se dice que un método es ágil cuando el desarrollo de software es:

- **Incremental:** Liberaciones pequeñas y ciclos rápidos.
- **Cooperativo:** Clientes y desarrolladores trabajando juntos.
- **Simple y Directo:** El método es fácil de aprender y modificar.
- **Adaptativo:** Es posible realizar cambios de último momento.

Algunas metodologías ágiles de desarrollo de software son:

- Adaptive Software Development (ASD).
- Agile Unified Process (AUP).
- Crystal Clear.
- Essential Unified Process (EssUP).
- Feature Driven Development (FDD).
- Lean Software Development (LSD).
- Programación Extrema (XP).
- Open Unified Process (OpenUp).
- Scrum.

3.2.1. Gestión ágil de proyectos

En la actualidad es necesario desarrollar y construir el producto a la par de la investigación y del descubrimiento de los requisitos, y hacerlo con la capacidad de adaptarse a los cambios dictados por el entorno.

El cliente conoce la visión de su producto pero por la novedad, el valor de innovación que necesita y la velocidad a la que se va a mover el escenario tecnológico y de negocio durante el desarrollo, no puede detallar cómo será el producto final.

La realidad insinúa que quizá ya no hay "productos finales", sino productos en evolución, revisión, mejora o incremento continuo a partir de la versión beta. En la figura 3.2 puede observarse un ejemplo de esta afirmación.

Figura 3.2. Ejemplo de un producto en evolución constante

La gestión ágil ha nacido de las prácticas empleadas en las empresas que mejor respuesta han sabido dar a las nuevas demandas. Los objetivos de la gestión ágil son:

- **Valor:** El principal valor del producto para las empresas que promueven el desarrollo ágil es *la innovación*. Se parte de la base de que la ingeniería concurrente genera mayor valor innovador. El solapamiento de las fases produce el modo de producción

que se ha venido a denominar *"ingeniería concurrente"*, en la que todas las personas implicadas en el proyecto, cliente incluido, trabajan de forma simultánea, en comunicación directa y combinando de forma simultánea toda la información del proyecto con el conocimiento y experiencia profesional de todo el equipo de desarrollo

- **Reducción de tiempo de desarrollo:** Esta reducción es un factor competitivo de primer orden para muchos productos. Las estrategias de la gestión ágil para producir resultados en menos tiempo que la gestión tradicional son:
  - *Entrega temprana de los primeros incrementos funcionales de producto*, que corresponden con las partes que con mayor urgencia necesita el cliente, de forma que puede lanzar la primera versión de producto en el menor tiempo posible.
  - *Solapamiento de las fases de desarrollo.*
- **Agilidad y flexibilidad**:
  - En la gestión ágil el objetivo es dar valor innovador. Los requisitos están abiertos, y se busca información en el propio avance del proyecto y en el entorno, de forma continua.
  - Capacidad de adaptación y evolución, a través de versiones, modificaciones, actualizaciones, ampliaciones, etc.
- **Fiabilidad:** Un modelo de gestión ágil es fiable si entrega de forma temprana y repetida un valor innovador.

La forma de gestión puede y debe adaptarse a las circunstancias del proyecto. La forma debe ser un buen acomodo, apropiado a la organización y al proyecto, para que puedan trabajar los principios que hacen funcionar el desarrollo ágil:
a) Operación preparada para responder al cambio, no para cumplir un plan.
b) Reducción al mínimo indispensable de las especificaciones documentadas como textos. Preferencia por prototipos ligeros (dibujos o representaciones de interfaz, simulaciones) o pesados (prototipos operativos o partes del sistema ya terminadas).
c) Implicación activa del cliente en el equipo de desarrollo.
d) Valor del conocimiento tácito de las personas y de su interacción por encima del conocimiento de los procesos.

3.2.2. Manifiesto Ágil

En marzo de 2001 diecisiete críticos de los modelos de mejora del desarrollo de software basados en procesos se reunieron para tratar sobre técnicas y procesos para desarrollar software, creando luego "*The Agile Alliance*", una organización, sin fines de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. Los integrantes de la reunión resumieron los principios sobre los que se basan los métodos alternativos en cuatro postulados, lo que ha quedado denominado como *Manifiesto Ágil*.

En el *Manifiesto Ágil* se expone que:

a) A los individuos y su interacción, por encima de los procesos y las herramientas.

b) El software que funciona, por encima de la documentación exhaustiva.

c) La colaboración con el cliente, por encima de la negociación contractual.

d) La respuesta al cambio, por encima del seguimiento de un plan.

A continuación, se explayará sobre cada uno de los puntos:

a) Valorar más a los individuos y su interacción que a los procesos y las herramientas

Este es posiblemente el principio más importante del manifiesto. Por supuesto que los procesos ayudan al trabajo. Son una guía de operación. Las herramientas mejoran la eficiencia, pero sin personas con conocimientos técnicos y actitud adecuada, no producen resultados.

Las empresas suelen predicar muy alto que sus empleados son lo más importante, pero la realidad es que en los años 90 la teoría de producción basada en procesos, la re-ingeniería

de procesos ha dado a éstos más relevancia de la que pueden tener en tareas que deben gran parte de su valor al conocimiento y al talento de las personas que las realizan.

Los procesos deben ser una ayuda y un soporte para guiar el trabajo. Deben adaptarse a la organización, a los equipos y a las personas; y no al revés. La defensa a ultranza de los procesos lleva a postular que con ellos se pueden conseguir resultados extraordinarios con personas mediocres, y lo cierto es que este principio es peligroso cuando los trabajos necesitan creatividad e innovación.

b) Valorar más el software que funciona que la documentación exhaustiva

Poder ver anticipadamente como se comportan las funcionalidades esperadas sobre prototipos o sobre las partes ya elaboradas del sistema final ofrece un "feedback" muy estimulante y enriquecedor que genera ideas imposibles de concebir en un primer momento; difícilmente se podrá conseguir un documento que contenga requisitos detallados antes de comenzar el proyecto.

El manifiesto no afirma que no hagan falta. Los documentos son soporte de la documentación, permiten la transferencia del conocimiento, registran información histórica, y en muchas cuestiones legales o normativas son obligatorios, pero se resalta que son menos importantes que los productos que funcionan. Menos trascendentales para aportar valor al producto.

Los documentos no pueden sustituir, ni pueden ofrecer la riqueza y generación de valor que se logra con la comunicación directa entre las personas y a través de la interacción con los prototipos. Por eso, siempre que sea posible debe preferirse, y reducir al mínimo indispensable el uso de documentación, que genera trabajo que no aporta un valor directo al producto.

Si la organización y los equipos se comunican a través de documentos, además de perder la riqueza que da la interacción con el producto, se acaba derivando a emplear a los documentos como barricadas entre departamentos o entre personas.

c) Valorar más la colaboración con el cliente que la negociación contractual

Las prácticas ágiles están especialmente indicadas para productos difíciles de definir con detalle en el principio, o que si se definieran así tendrían al final menos valor que si se van enriqueciendo con retro-información continua durante el desarrollo. También para los casos en los que los requisitos van a ser muy inestables por la velocidad del entorno de negocio.

Para el desarrollo ágil el valor del resultado no es consecuencia de haber controlado una ejecución conforme a procesos, sino de haber sido implementado directamente sobre el producto. Un contrato no aporta valor al producto. Es una formalidad que establece líneas divisorias entre responsabilidades, que fija los referentes para posibles disputas contractuales entre cliente y proveedor.

En el desarrollo ágil el cliente es un miembro más del equipo, que se integra y colabora en el grupo de trabajo. Los modelos de contrato por obra no encajan.

d) Valorar más la respuesta al cambio que el seguimiento de un plan

Para un modelo de desarrollo que surge de entornos inestables, que tienen como factor inherente al cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que la de seguimiento y aseguramiento de planes pre-establecidos. Los principales valores de la gestión ágil son la anticipación y la adaptación; diferentes a los de la gestión de proyectos ortodoxa: planificación y control para evitar desviaciones sobre el plan.

Principios del Manifiesto Ágil
Tras los cuatro valores descritos, los firmantes redactaron los siguientes, como los principios que de ellos se derivan:

- Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
- Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
- Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.
- Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
- Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la principal medida del progreso.
- Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica enaltece la agilidad.
- La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.
- En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

3.2.3. Comparación entre las "Metodologías Tradicionales" y los "Procesos Ágiles"

Hay dos formas de gestionar proyectos: ***una predictiva o clásica y otra adaptativa o ágil***.

La gestión predictiva parte de un plan detallado. Sabe exactamente qué es lo que va a hacer y conoce fechas y costes. Durante el desarrollo gestiona riesgos, evalúa el impacto que cada modificación supone sobre el plan inicial, toma decisiones frente a los imprevistos para seguir su cumplimiento y, si no queda más remedio, para replantearlo. Este modelo busca conseguir previsibilidad: construir lo previsto, en el tiempo previsto y con el coste previsto. Hay gestores que sólo trabajan de forma predictiva por la creencia fundamentalista de que es "la buena", y descalifican a los enfoques ágiles o adaptativos.

Las premisas sobre las que se desarrolló la gestión de proyectos tradicional fueron:
a) Todos los proyectos mantienen características y comportamientos regulares (Norden, Julio 1958).
b) El objetivo de la ejecución de un proyecto es lograr el producto previsto en el tiempo planificado sin desbordar los costes estimados.

Y es en base a las dos premisas anteriores, que se determinaron estas características:
a. ***Universalidad:*** Los proyectos, pese a su diversidad, comparten patrones comunes de ejecución y regularidad. Las prácticas de gestión trabajan sobre esos patrones comunes y resultan válidas para cualquier tipo de proyecto.
b. ***Carácter predictivo:*** La gestión predictiva define con detalle cuál es el producto previsto y elabora un plan de desarrollo sobre el que calcula costes y fechas. Durante la ejecución realiza actividades de seguimiento y vigilancia para evitar desviaciones sobre lo planificado.

La gestión adaptable parte desde una visión general del objetivo y va dando pequeños pasos hacia él a través de un ciclo de construcción incremental, y de forma evolutiva contrasta y va descubriendo el detalle que resulta más adecuado para el producto con los usuarios y demás participantes, que pueden "tocar" o usar las partes construidas. Este modelo busca conseguir valor competitivo en entornos rápidos de la economía actual: innovación y agilidad. Hay gestores que sólo trabajan de forma adaptativa por la creencia fundamentalista de que es "la buena", y descalifican a los enfoques clásicos o predictivos.

En las figuras 3.3. y 3.4. se muestran algunas diferencias entre los dos tipos de metodologías.

Metodologías Ágiles	**Metodologías Tradicionales**
Equipo multidisciplinar	Especialización por fase
Solapamiento de Fases	Fases dependientes
Visión del producto	Requisitos detallados
Adaptación a los cambios	Seguimiento del plan

Figura 3.3. Diferencias generales entre metodologías tradicionales y ágiles

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Figura 3.4. Diferencias entre metodologías tradicionales y ágiles

Finalmente en la figura 3.5. se muestran diferencias en los procesos de desarrollos.

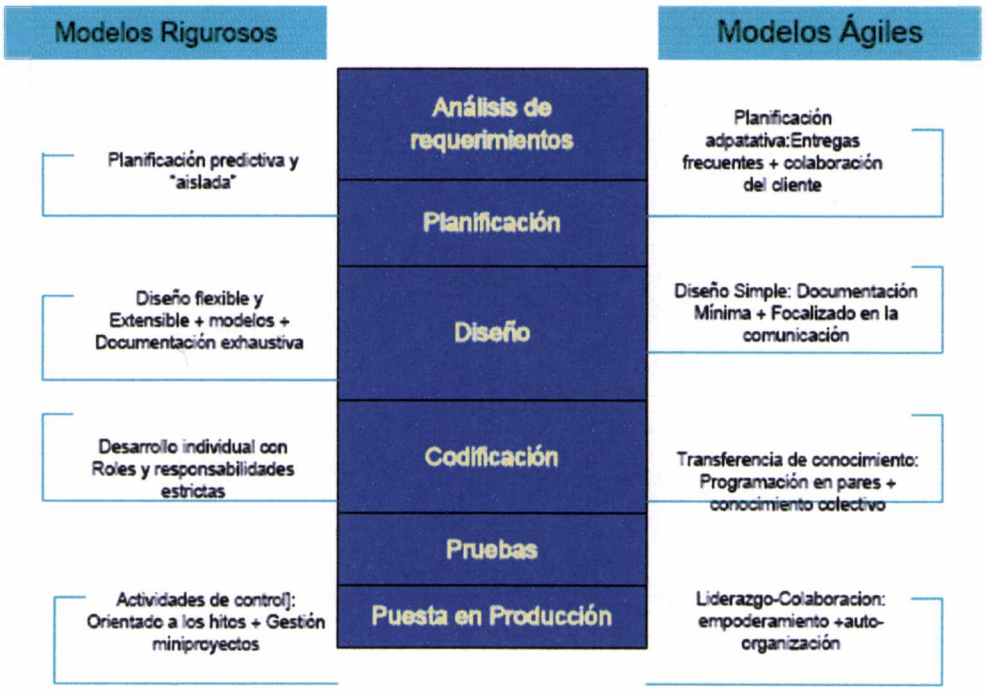

Figura 3.5. Diferencias de los procesos de desarrollo entre metodologías tradicionales y ágiles

3.2.3.1. Algunas desventajas y restricciones de las Metodologías Tradicionales

- Las transiciones entre fases, acaban funcionando como fronteras. Cada equipo se siente responsable de su parte de trabajo, de lo que debe entregar a la siguiente fase, pero no del resultado final.
- Los documentos de diseño, los requisitos o los prototipos pueden acabar siendo barricadas en la frontera de cada fase, que lejos de favorecer a la comunicación directa fomentan la fragmentación.

- Los retrasos de cada fase son cuellos de botella del proyecto. El solapamiento diluye el ruido y los problemas entre fases.
- Se requieren personas/equipos especializados para cada una de las fases.

3.2.3.2. Algunas desventajas y restricciones de las Metodologías Ágiles

- Están dirigidas a equipos pequeños o medianos (Se sugiere que el tamaño de los equipos se limite de 3 a 10 personas; algunos sugieren que podrían soportar 20 personas como máximo)
- El entorno físico debe ser un ambiente que permita la comunicación y colaboración entre todos los miembros del equipo durante todo el tiempo
- Cualquier resistencia del cliente o del equipo de desarrollo hacia las prácticas y principios puede llevar al proceso al fracaso (el clima de trabajo, la colaboración y la relación contractual son claves),
- El uso de tecnologías que no tengan un ciclo rápido de realimentación o que no soporten fácilmente el cambio
- Se requieren de personas que tengan conocimientos multidisciplinares.
- Al tener escasa documentación técnica se dificulta la transferencia de un proyecto a otro grupo

3.2.3.3. ¿Cuando utilizar metodologías tradicionales y cuando utilizar procesos ágiles?

Es cierto que muchas características que diferencian unos proyectos de otros son superficiales y resultan indiferentes para el modelo de gestión; pero hay otras que permiten adoptar prácticas de gestión muy distintas.

Además de las dos premisas sobre las que se desarrolló la gestión de proyectos tradicional, existen otras dos que son las más cuestionadas:

a) **El objetivo es producir el producto definido, en costes y fechas:** ¿El objetivo de cualquier proyecto siempre es: producto, costes y fechas planificadas? ¿Y si las necesidades de innovación y agilidad pesan más que las de previsibilidad? ¿Y si el cliente no sabe o no quiere dar un dibujo cerrado del producto; no quiere que le cierren los requisitos, y prefiere incluir modificaciones y cambios de forma continua? ¿Y si no se trata de acotar el marco temporal de desarrollo, sino de tener el producto en continuo desarrollo, dándole valor de forma constante?

b) **Todos los proyectos comparten los mismos patrones de ejecución:** ¿Todos los proyectos comparten los mismos patrones de ejecución? ¿Es adecuado emplear los mismos principios en dos proyectos tan diferentes, como el desarrollo de un sistema para aeropuertos y un foro Web?

Hay características relevantes que piden estrategias de gestión de proyectos diferentes. Pueden ser:

- Por las necesidades del cliente
- Por las características del proyecto
- Por las características de la organización que desarrolla el sistema

Las características determinantes para la gestión de un proyecto, involucran, al proyecto, al cliente y a la organización. En la figura 3.6 se enumeran cada uno de los factores a considerar.

Cliente	Prioridad de negocio
Proyecto	Estabilidad de los requisitos
	Maleabilidad y coste de la materia prima
	Criticidad del sistema que se debe construir
	Coste de prototipado
	Tamaño del equipo
Organización	Cultura de la organización
	Nivel técnico del equipo
	Estrategia de desarrollo: procesos / personas

Figura 3.6. Características determinantes para la gestión de proyectos

Para determinar el tipo de metodología a utilizar es necesario realizarse ciertos cuestionamientos:

- **Cliente**
  - Prioridad de negocio: *¿Qué tiene más importancia: el cumplimiento de agendas y fechas o el valor innovador del producto?* La gestión predictiva es un modelo especializado en el cumplimiento de planes. La gestión adaptable es un modelo especializado en dotar al producto del mayor valor innovador posible.
- **Proyecto**
  - Estabilidad de los requisitos: *¿Se puede obtener una descripción de requisitos detallada al inicio del proyecto?, y ésta, ¿se mantendrá estable durante el desarrollo?*
  - Maleabilidad y coste de la materia prima: ¿Cuán fácil es modificar el producto?
  - Coste del prototipado: El coste de prototipar podría verse como una característica independiente de la maleabilidad, pero suelen estar en estrecha relación, y en el caso del software, ambas son prácticamente la misma cosa.
  - Criticidad del sistema que se debe construir: Si el sistema falla, se degrada o no consigue realizar las funciones de los requisitos, *¿qué impacto tiene en la seguridad o en el rendimiento?* (¿Causará daño a personas? ¿Causará daño al medio ambiente? ¿Producirá pérdidas económicas graves? ¿Producirá pérdidas económicas? ¿Fallará la finalidad principal del sistema? ¿Fallarán funcionalidades auxiliares del sistema? ¿Se producirán fallos ergonómicos o de comodidad para los usuarios?)
  - Tamaño del equipo: ¿Cuan grande es el equipo de desarrollo?
- **Organización**
  - Cultura organizativa: Para la ejecución sistemática y controlada de procesos no resulta especialmente relevante el tipo de cultura de la organización. Para el desarrollo de trabajo basado en el talento de las personas resultan inhibidores los ambientes laborales basados en el control, excesivamente normalizados y jerarquizados.
  - Nivel profesional: Si el proyecto, más que innovación lo que requiere es ejecución controlada de un plan detallado, posiblemente sean los procesos de la organización los garantes del resultado, y con un modelo de gestión predictiva, el factor relevante es más la capacidad de los procesos empleados que un elevado nivel profesional de las personas del equipo. Si por ser el valor del producto el objetivo del proyecto se emplea un modelo de desarrollo ágil, son las personas y no los procesos los encargados de proporcionarlo y en ese caso el equipo debe estar compuesto por personas con el mayor conocimiento y experiencia posible.
  - Modelo de desarrollo: Los entornos de desarrollo basados en procesos son adecuados para modelos de gestión predictiva. Los entornos de desarrollo basados en las personas son adecuados para modelos de gestión ágil.

3.3. OpenUp

OpenUp abarca una filosofía ágil y pragmática que se centra en el desarrollo de software colaborativo. Es un proceso que puede ser extendido para abordar una amplia variedad de tipos de proyectos.

Es un proceso de desarrollo de software mínimo ya que solo abarca contenidos básicos. Por lo tanto no proporciona orientación sobre muchos temas que los proyectos pueden abordar como los equipos grandes, el cumplimiento, las situaciones contractuales, aplicaciones críticas, etc.

De todas maneras, OpenUp es completo en el sentido de que puede manifestarse como un proceso para construir un sistema. Es una metodología extensible que puede adaptarse según sea necesario.

En necesario aclarar que OpenUp está en el limite entre ser una metodología ágil y una tradicional, ya que es un método que deriva de RUP. De todos modos en las bibliografías consultadas se la considera una metodología ágil, aun discutiendo este punto.

3.3.1. Principios de OpenUp

OpenUp tiene las características esenciales de un proceso unificado que aplica de forma iterativa e incremental el ciclo de vida estructurado. Está basado en casos de uso, la gestión del riesgo, y una arquitectura de un enfoque centrado en impulsar el desarrollo.

OpenUp es impulsada por cuatro principios fundamentales que se citan a continuación:

- Colaborar para sincronizar intereses y compartir conocimiento. Este principio promueve prácticas que impulsan un ambiente de equipo saludable, facilitan la colaboración y desarrollan un conocimiento compartido del proyecto.
- Equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto. Este principio promueve prácticas que permiten a los participantes de los proyectos desarrollar una solución que maximice los beneficios obtenidos por los participantes y que cumple con los requisitos y restricciones del proyecto.
- Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo.
- Desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo. Este principio promueve prácticas que permiten a los equipos de desarrollo obtener retroalimentación temprana y continua de los participantes del proyecto, permitiendo demostrarles incrementos progresivos en la funcionalidad.

Cada principio OpenUp apoya una declaración en el Manifiesto ágil, como se ve en la figura 3.7.

Principios OpenUp	Declaración del Manifiesto ágil
Colaborar para sincronizar intereses y compartir conocimiento	A los individuos y su interacción por encima de los procesos y las herramientas
Equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto	Colaboración con el cliente por encima de la negociación contractual
Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo.	Software que funciona por encima de la documentación exhaustiva
Desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo	La respuesta del cambio por encima del seguimiento de un plan.

Figura 3.7. Comparación entre OpenUp y el "Manifiesto Ágil"

3.3.2. Elementos de OpenUp

Los elementos que forman OpenUp se pueden enumerar en:

- **Disciplinas:** OpenUp se centra en las siguientes disciplinas: Requisitos, arquitectura, desarrollo, pruebas, gestión de proyectos, y configuración y cambios en la gestión. Otras disciplinas fueron omitidas, como el modelo de negocio, el manejo de

requerimientos avanzado entre otras. Estas son consideradas innecesarias para un proyecto pequeño o son manejados por otras áreas de la organización, fuera del equipo del proyecto.

- **Tareas:** En OpenUp se define tarea como la unidad de trabajo que debe ser realizada por algún rol. En OpenUp se cuenta con un total de 18 tareas que son ejecutadas de forma principal por un rol (el encargado de realizar la tarea) y de forma secundaria por roles que apoyan la ejecución de dicha tarea. La naturaleza colaborativa de OpenUp queda demostrada al observarse que ninguna tarea es ejecutada de forma aislada por un único rol sin el apoyo de ningún otro.
- **Artefactos:** Un artefacto en OpenUp se considera a todo aquello que una tarea necesita para realizar su función, o bien lo produce o modifica. Los distintos roles existentes son los encargados de crear y actualizar los artefactos. En OpenUp existen un total de 17 artefactos predefinidos considerados como esenciales, pero lógicamente la organización puede decidir añadir tantos artefactos propios como considere necesario, incluso se puede optar por ignorar estos artefactos predefinidos y utilizar cualquier tipo de artefactos definidos por la organización, siempre y cuando estos cumplan el objetivo para el cual fueron creados: almacenar información. Generalmente los artefactos suelen estar regulados por un sistema de control de versiones.
- **Procesos:** Los procesos toman los elementos metodológicos y los relacionan entre si dentro de secuencias temporales que satisfacen las necesidades de distintos tipos de proyecto. Aquellos elementos metodológicos con un alto grado de reutilización se agrupan en patrones de capacitación, aportando un enfoque de desarrollo consistente para la mayoría de proyectos de desarrollo. Estos patrones se realizan organizando las tareas en actividades, agrupándolas dentro de una secuencia que tiene sentido para un área en particular en la que el patrón es aplicado. Algunos de estos proyectos pueden usarse como plantillas para alguna de las iteraciones del proyecto.

3.3.3. Roles en un proyecto OpenUp

- **Stakeholder**: Representa grupos interesados a los que debe satisfacer el proyecto. Es un rol que puede jugar por cualquier persona que es (o será potencialmente) significativamente afectada por los resultados del proyecto.
- **Analyst**: Representan las inquietudes del cliente y usuario final mediante reuniones con las partes interesadas para entender el problema a resolver, mediante el reconocimiento y captura de los requisitos principales.
- **Architect:** Es responsable de diseñar la arquitectura de software, que incluye la toma de decisiones técnicas claves que limitan el diseño y la ejecución general del proyecto.
- **Developer:** Es responsable de desarrollar una parte del sistema, incluyendo el diseño para ajustarse a la arquitectura, y luego implementar test de unidad e integrar los componentes como parte de la solución.
- **Tester:** Es responsable de las actividades básicas de prueba, como la identificación, definición, implementación y realización de las pruebas necesarias, así como el registro de los resultados de las pruebas y análisis de los resultados.
- **Project Manager:** Lidera la planificación del proyecto en colaboración con las partes interesadas y el equipo, coordina las interacciones con los interesados, y mantiene al equipo del proyecto enfocado en el cumplimiento de los objetivos del proyecto.
- **Any Role:** Es un integrante del equipo que realiza tareas generales

3.3.4. Fases de un proyecto en OpenUp

Los integrantes del equipo contribuyen aportando microincrementos que puede ser el resultado del trabajo de unas pocas horas o unos pocos días. El progreso se puede visualizar diariamente, ya que la aplicación va evolucionando en función de estos microincrementos.

Organización de OpenUp

OpenUp está organizado en 2 diferentes dimensiones relacionadas: El contenido del método y el contenido del proceso. El contenido método es donde se definen los elementos del método (es decir, roles, tareas, artefactos, y orientación), son definidos independientemente de la forma en que se utilizan en un ciclo de vida del proyecto. El contenido de los procesos es donde los elementos del método se aplican temporalmente.

Desde el mismo conjunto de elementos del método se pueden crear varios ciclos de vida para diferentes proyectos

Los elementos de OpenUp dirigen la organización del trabajo en los niveles personal, de equipo y de interesados.

A nivel personal, los integrantes de un proyecto contribuyen con su trabajo con pequeños incrementos en funcionalidad, denominados microincrementos, los cuales representan los resultados obtenidos en pocas horas o pocos días de trabajo. La solución evoluciona basada en dichos microincrementos de tal forma que el progreso puede ser visualizado efectivamente cada día. Los integrantes del equipo de desarrollo de forma abierta comparten su progreso diario el cual incrementa la visibilidad en el trabajo, la confianza y el trabajo en equipo.

El proyecto en general se divide en iteraciones, las cuales son planificadas en un intervalo definido de tiempo que no superan las pocas semanas. El OpenUp tiene elementos que ayudan a los equipos de trabajo a enfocar los esfuerzos a través del ciclo de vida de cada iteración de tal forma que se puedan distribuir funcionalidades incrementales de una manera predecible, una versión totalmente probada y funcional al final de cada iteración.

En la figura 3.8. se muestra el ciclo de vida de un proyecto en la metodología OpenUp en cada una de sus dimensiones.

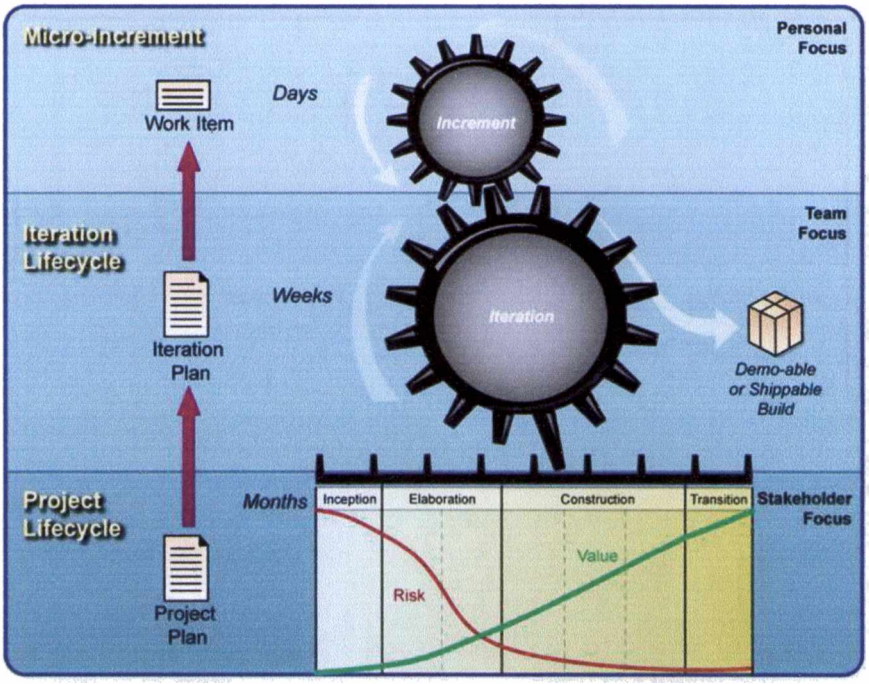

Figura 3.8. Estructura de las fases de OpenUp

El objetivo de OpenUp es ayudar al equipo de desarrollo a través de todo el ciclo de vida de las iteraciones, de modo que este sea capaz de añadir valor de negocio para los clientes de una forma predecible: con la entrega de un software operativo y funcional al final de cada iteración. El ciclo de vida del proyecto provee a los clientes de una visión del proyecto, transparencia y les dota de los medios para que les permitan controlar la financiación, el riesgo, el ámbito, el valor de retorno esperado, etc.

Todo proyecto en OpenUp consta de cuatro fases: Inicio, Elaboración, Construcción y Transición. Cada una de estas fases se divide a su vez en iteraciones cada una de las cuales tiene como objetivo la entrega de un software operativo y funcional.

A continuación se explica cada una de las fases:

- **Fase de inicio:** Las necesidades de cada participante del proyecto son tenidas en cuenta y son plasmadas en objetivos del proyecto. Se deben definir el ámbito del proyecto, los limites del mismo y el criterio de aceptación del proyecto. Los casos de uso críticos, aquellos que dirigen la funcionalidad del sistema, son definidos en esta fase, así como una estimación inicial del coste del proyecto y un boceto de la planificación.
- **Fase de elaboración:** En esta fase se realizan tareas de análisis del dominio y definición de la arquitectura del sistema. Si se decide continuar con el proyecto se debe elaborar un plan de proyecto en esta fase, para lo cual se deben establecer unos requisitos y arquitectura estables. Por otro lado el proceso de desarrollo, las herramientas, la infraestructura a utilizar y el entorno de desarrollo también se especifican en detalle en esta fase. Al final de la fase se debe tener una definición clara y precisa de los casos de uso, los actores, la arquitectura del sistema y un prototipo ejecutable de la misma.
- **Fase de construcción:** Todos los componentes y funcionalidades del sistema que falten por implementar son realizados, testeados e integrados en esta fase. Los resultados obtenidos en forma de incrementos ejecutables deben ser desarrollados de la forma más rápida posible sin dejar de lado la calidad de lo desarrollado.
- **Fase de transición:** Cuando el producto esta lo suficientemente maduro (algo que es establecido por ejemplo en función del numero de peticiones de cambio por parte de los clientes) como para ser introducido en la comunidad de usuarios, el proyecto se encuentra en esta fase. Las fases de la transición constan de subfases de testeo de versiones beta, pilotaje y capacitación de los usuarios finales y de los encargados del mantenimiento del sistema. En función de la respuesta obtenida por los usuarios puede que haya que realizar cambios en las entregas finales o implementar alguna funcionalidad más.

En la figura 3.9. se puede ver con más detalle una solución al proceso.

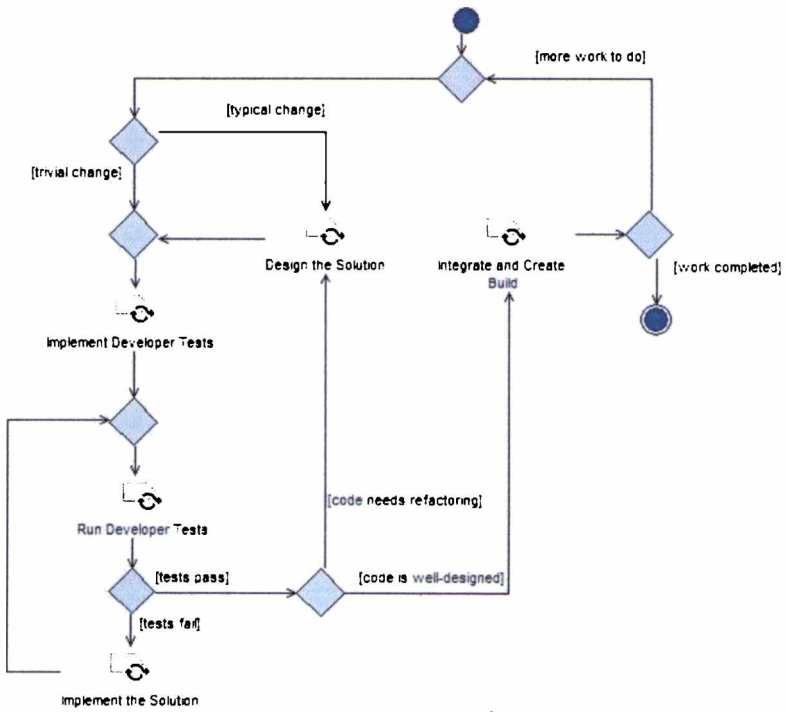

Figura 3.9. Ejemplo de solución OpenUp

3.3.5. Prácticas en OpenUp

OpenUp es una metodología basada en RUP, y por lo tanto, comparte las mismas prácticas que subyacen por debajo del flujo de trabajo y los roles de OpenUp. Estas prácticas son:

- **Desarrollar software de manera iterativa:** Todo desarrollo debe realizarse de modo iterativo y en pequeños incrementos. Esto permite identificar riesgos de forma prematura, y actuar en consecuencia.
- **Gestionar los requisitos:** La identificación de los requisitos clave y los que cambian continuamente es un proceso de que debe ejecutarse continuamente. Deben articularse los medios que permitan priorizar, filtrar y hacer un seguimiento de los requisitos, ya que, la comunicación será mucho mas eficaz si esta basada en unos requisitos claramente definidos.
- **Hacer uso de una arquitectura basada en componentes:** Las arquitecturas basadas en componentes son más flexibles, y dado que OpenUp es una metodología ágil, y por lo tanto flexible, la arquitectura definida debe estar basada en componentes. Por otro lado, la construcción de componentes con un alto grado de reutilización puede ahorrar considerable esfuerzo de futuros desarrollos.
- **Modelizar software de forma visual:** Se deben aplicar métodos de visualización de modelos que permitan entender la complejidad del sistema sin dar cabida a la ambigüedad.
- **Verificar la calidad del software:** Si se realizan verificaciones en cada iteración de cada una de las fases, se reduce considerablemente el costo de resolver cualquier defecto, ya que estos se identifican prematuramente.
- **Controlar los cambios aplicados al software:** Todo cambio que se solicite sobre los requisitos establecidos se debe gestionar y realizar un seguimiento sobre el mismo, de este modo se puede determinar el grado de madurez del software desarrollado.

3.4. Scrum

Pese a ser un proceso que surgió como modelo en el desarrollo de productos tecnológicos, sus principios son válidos para entornos que trabajan con requisitos inestables, y necesitan agilidad: situaciones frecuentes en el desarrollo de determinados sistemas de software.

Desde sus principios, el nuevo modelo de desarrollo no se trata de un cierto solapamiento entre fases, sino de un solapamiento tan amplio que durante la totalidad del desarrollo concurren todas las actividades.

De esta forma, más que fases que se realizan de forma secuencial, pasan a ser actividades que se ejecutan en el momento que se requieren. Requisitos, análisis, codificación, pruebas, integración se van realizando en cada momento según las necesidades en la evolución del proyecto. A este entorno de trabajo se lo denominó "*campo de Scrum*", por la analogía entre el equipo de trabajo y un equipo de rugby.

Aludiendo explícitamente al desarrollo de software, puede describirse a Scrum como es una metodología de desarrollo muy simple, que requiere trabajo duro, porque la gestión no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto. Al tratarse de una metodología ágil puede afirmarse que:

- Es un modo de desarrollo de carácter adaptable.
- Orientado a las personas antes que a los procesos.
- Emplea desarrollo ágil: iterativo e incremental.

La denominación "*campo de Scrum*" surgió por estudios realizados sobre nuevas prácticas de producción en diversas empresas (Nonaka y Takeuchi, 1986). Las características comunes que se identificaron en los entornos de desarrollo de las empresas analizadas fueron:

- ***La incertidumbre como elemento consustancial y asumido en el entorno y en la cultura de la organización:*** En estas empresas, desde la dirección se apunta cuál es la visión genérica que se quiere conseguir, o la dirección estratégica que hay que seguir, pero no un plan detallado del producto y su desarrollo. Al mismo tiempo se da al equipo un margen de libertad amplio. Los ingredientes clave que sirven para la creatividad y compromiso del equipo son:
  - La "tensión" que crea la visión difusa y el reto que supone el grado de dificultad que encierra.
  - El margen de autonomía, libertad y responsabilidad.
- ***Equipos de desarrollo auto-organizados:*** No hay roles de gestión que marquen pautas o asignación de tareas. Para que los equipos puedan conseguir auto-organizarse deben reunir tres características:
  - Autonomía: son libres para elegir la estrategia de solución.
  - Auto-superación. El equipo va desarrollando soluciones que evalúa analiza y mejora.
  - Auto-enriquecimiento: La multidisciplinaridad de los componentes del equipo favorece el enriquecimiento mutuo y la adopción de soluciones valiosas y complementarias.
- ***Fases de desarrollo solapadas:*** En el desarrollo ágil las "fases" pasan a ser "actividades". El concepto de fase implica sucesión secuencial de unas a otras. En un campo de Scrum los trabajos que se llevan a cabo pierden el carácter de fase y son actividades que se realizan en cualquier momento, de forma simultánea, o a demanda según las necesidades en cada iteración. La fase de requisitos ya se hizo; ya está completada.
- ***Control sutil:*** El equipo trabaja con autonomía en un entorno de ambigüedad, inestabilidad y tensión. La gestión establece puntos de control suficientes para evitar que el ambiente de ambigüedad, inestabilidad y tensión del "campo de Scrum" derive hacia descontrol. Las acciones para generar el ecosistema de este control son:
  - Seleccionando a las personas adecuadas para el proyecto, y analizando los cambios en la dinámica del grupo para incorporar o retirar a personas si resulta necesario.
  - Creando un espacio de trabajo abierto.
  - Animando a los ingenieros a "mezclarse" con el mundo real de las necesidades de los clientes.
  - Estableciendo sistemas de evaluación y reconocimiento basados en el rendimiento del equipo.
  - Gestionando las diferencias de ritmo a través del proceso de desarrollo.
  - Siendo tolerante y previsor con los errores: son un medio de aprendizaje, y el miedo al error merma la creatividad y la espontaneidad.
  - Implicando a los proveedores en el proyecto y animándoles también a su propia auto-organización
- ***Difusión y transferencia del conocimiento:*** Tanto a nivel de proyecto como de organización. Los equipos son multidisciplinares; todos los miembros aportan y aprenden tanto del resto del equipo como de las investigaciones, innovaciones de su producto y de la experiencia del desarrollo. Las personas que participan en un proyecto, con el tiempo van cambiando de equipo en la organización, a otros proyectos; de esta forma se van compartiendo y comunicando las experiencias en la organización. Los equipos y las empresas mantienen libre acceso a la información, herramientas y políticas de gestión del conocimiento

El desarrollo se inicia desde la visión general de producto, dando detalle solo a las funcionalidades que, por ser las de mayor prioridad para el negocio, se van a desarrollar en primer lugar, y pueden llevarse a cabo en un periodo de tiempo breve (entre 15 y 60 días).

Cada uno de los ciclos de desarrollo es una iteración (sprint) que produce un incremento terminado y operativo del producto. Estas iteraciones son la base del desarrollo ágil, y Scrum gestiona su evolución a través de reuniones breves de seguimiento en las que todo el equipo revisa el trabajo realizado desde la reunión anterior y el previsto hasta la reunión siguiente. Se aconseja que las reuniones de seguimiento del sprint (iteración) sean diarias.

Scrum es un proceso en el que se aplican de manera regular un conjunto de mejores prácticas para trabajar en equipo y obtener el mejor resultado posible de un proyecto; Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos. En Scrum se realizan entregas parciales y regulares del resultado final del proyecto, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad y la productividad son fundamentales.

Scrum es un proceso de desarrollo de software iterativo y creciente utilizado comúnmente en entornos basados en el desarrollo ágil de software.

A modo sintético, en la figura 3.10. puede verse que la metodología resulta sencilla definiendo algunos roles y artefactos que contribuyen a tener un proceso que maximiza el feedback para mitigar cualquier riesgo que pueda presentarse.

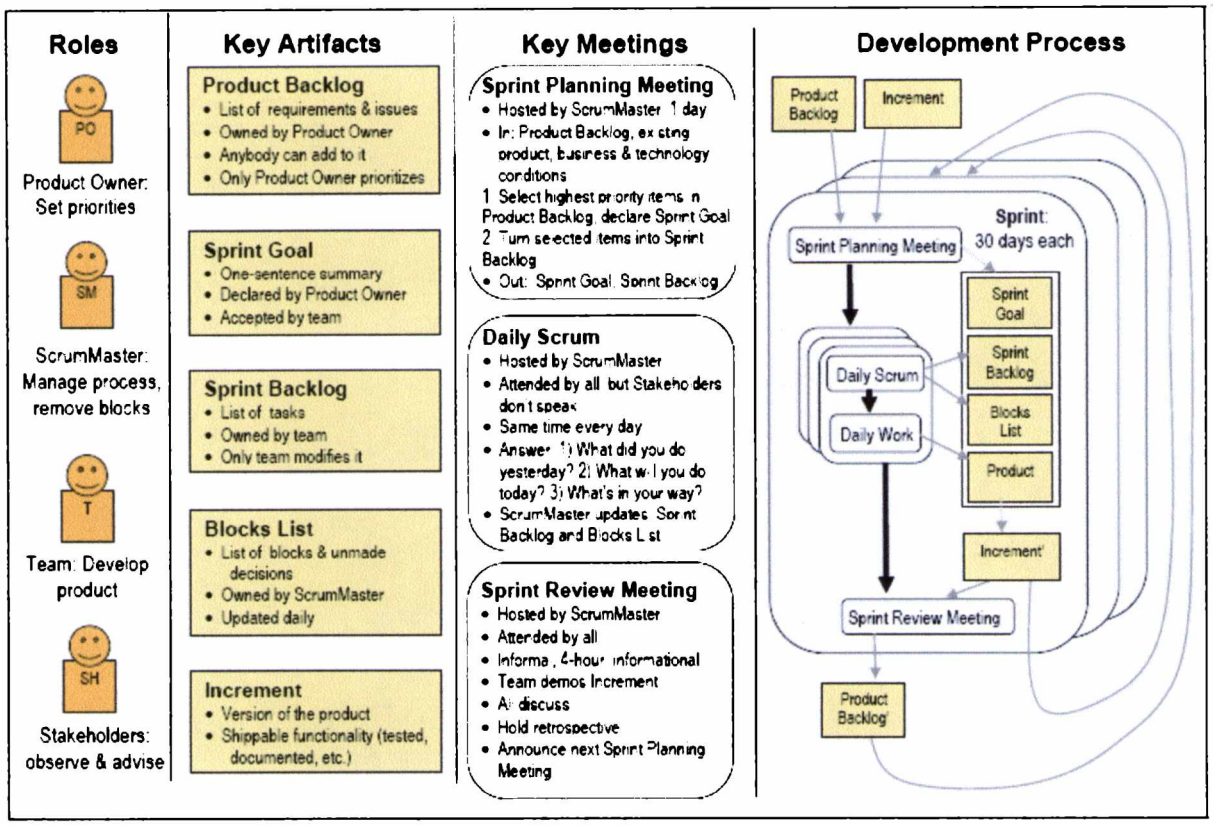

Figura 3.10. Ejemplo de Scrum

Fundamentos de Scrum
Scrum se basa en:

- El desarrollo incremental de los requisitos del proyecto en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas, si así se necesita).
- La priorización de los requisitos por valor para el cliente y coste de desarrollo en cada iteración.
- El control empírico del proyecto. Por un lado, al final de cada iteración se demuestra al cliente el resultado real obtenido, de manera que pueda tomar las decisiones necesarias en función de lo que observa y del contexto del proyecto en ese momento. Por otro lado, el equipo se sincroniza diariamente y realiza las adaptaciones necesarias.
- La potenciación del equipo, que se compromete a entregar unos requisitos y para ello se le otorga la autoridad necesaria para organizar su trabajo.
- La sistematización de la colaboración y la comunicación tanto entre el equipo, como con el cliente.
- El timeboxing de las actividades del proyecto, para forzar la toma de decisiones y conseguir resultados.

3.4.1. Actores y roles en Scrum

En Scrum se definen varios roles, estos están divididos en dos grupos: *cerdos y gallinas*. Los nombres de los grupos están inspirados en el chiste sobre un cerdo y una gallina que se relata a continuación.

> *Un cerdo y una gallina se encuentran en la calle. La gallina mira al cerdo y dice: "Hey, ¿por qué no abrimos un restaurante?" El cerdo mira a la gallina y le dice: "Buena idea, ¿cómo se llamaría el restaurante?" La gallina piensa un poco y contesta: "¿Por qué no lo llamamos "Huevos con jamón?" "Lo siento pero no", dice el cerdo, "Yo estaría comprometido pero tú solamente estarías involucrada".*

De esta forma, los *cerdos* están comprometidos a construir software de manera regular y frecuente, mientras que el resto son *gallinas*: interesados en el proyecto pero realmente irrelevantes porque, si éste falla, no son un *cerdo*, es decir, no son los que se habían comprometido a sacarlo adelante. Las necesidades, deseos, ideas e influencias de los roles *gallina* se tienen en cuenta, pero no de forma que pueda afectar, distorsionar o entorpecer el proyecto Scrum.

3.4.1.1. Roles "Cerdo"

Los *Cerdos* son los que están comprometidos con el proyecto y el proceso Scrum. Entre estos roles se encuentran:

- ***Product Owner: Las responsabilidades del Product Owner (que puede ser interno o externo a la organización) son:***
  - Ser el representante de todas las personas interesadas en los resultados del proyecto (internas o externas a la organización, promotores del proyecto y usuarios finales [idealmente también debería ser un usuario clave] o consumidores finales del producto) y actuar como interlocutor único ante el equipo, con autoridad para tomar decisiones.
  - Definir los objetivos del producto o proyecto.
  - Dirigir los resultados del proyecto y maximizar su ROI (*Return Of Investment*).
    - Es el propietario de la planificación del proyecto: crea y mantiene la lista priorizada con los requisitos necesarios para cubrir los objetivos del producto o proyecto, conoce el valor que aportará cada requisito y calcula el ROI a partir del coste de cada requisito que le proporciona el equipo.
    - Reparte los objetivos/requisitos en iteraciones y establece un calendario de entregas.
    - Antes de iniciar cada iteración replanifica el proyecto en función de los requisitos que aportan más valor en ese momento, de los requisitos completados en la iteración anterior y del contexto del proyecto en ese momento (demandas del mercado, movimientos de la competencia, etc.).
  - Participar en la reunión de planificación de iteración, proponiendo los requisitos más prioritarios a desarrollar, respondiendo a las dudas del equipo y detallando los requisitos que el equipo se comprometer a hacer.
  - Estar disponible durante el curso de la iteración para responder a las preguntas que puedan aparecer.
  - No cambiar los requisitos que se están desarrollando en una iteración, una vez está iniciada.
  - Participar en la reunión de demostración de la iteración, revisando los requisitos completados.
- ***ScrumMaster: Lidera al equipo llevando a cabo las siguientes responsabilidades:***
  - Velar porque todos los participantes del proyecto sigan las reglas y el proceso Scrum, encajándolas en la cultura de la organización, y guiar la colaboración intraequipo y con el cliente de manera que las sinergias sean máximas. Esto implica:
    - Asegurar que la lista de requisitos priorizada esté preparada antes de la siguiente iteración.
    - Facilitar las reuniones de Scrum (planificación de la iteración, reuniones diarias de sincronización del equipo, demostración, retrospectiva), de manera que sean productivas y consigan sus objetivos.
    - Enseñar al equipo a autogestionarse. No da respuestas, si no que guía al equipo con preguntas para que descubra por sí mismo una solución.
  - Quitar los impedimentos que el equipo tiene en su camino para conseguir el objetivo de cada iteración (proporcionar un resultado útil al cliente de la manera más efectiva) y poder finalizar el proyecto con éxito. Estos obstáculos se identifican de manera sistemática en las reuniones diarias de sincronización del equipo y en las reuniones de retrospectiva.

- Proteger y aislar al equipo de interrupciones externas durante la ejecución de la iteración (introducción de nuevos requisitos, "secuestro" no previsto de un miembro del equipo, etc.). De esta manera, el equipo puede mantener su productividad y el compromiso que adquirió sobre los requisitos que completaría en la iteración (notar, sin embargo, que el equipo debe reservar tiempo para colaborar con al cliente en la preparación de la lista de requisitos para la próxima iteración).
- Asegurar que los requisitos se desarrollan con calidad.

- ***Team (Equipo): Grupo de personas que de manera conjunta desarrollan el producto del proyecto.***
  - Tienen un objetivo común, comparten la responsabilidad del trabajo que realizan (así como de su calidad) en cada iteración y en el proyecto.
  - El tamaño del equipo está entre 5 y 9 personas.
  - Es un equipo autoorganizado, que comparte información y cuyos miembros confían entre ellos.
  - Realiza de manera conjunta las siguientes actividades:
    - Seleccionar los requisitos que se compromete a completar en una iteración, de forma que estén preparados para ser entregados al cliente.
    - Estimar la complejidad de cada requisito en la lista de requisitos priorizada del producto o proyecto.
    - En la reunión de planificación de la iteración decide cómo va a realizar su trabajo:
      - Seleccionar los requisitos que pueden completar en cada iteración, realizando al cliente las preguntas necesarias.
      - Identificar todas las tareas necesarias para completar cada requisito.
      - Estimar el esfuerzo necesario para realizar cada tarea.
      - Cada miembro del equipo se autoasigna a las tareas.
    - Durante la iteración, trabajar de manera conjunta para conseguir los objetivos de la iteración. Cada especialista lidera el trabajo en su área y el resto colaboran si es necesario para poder completar un requisito.
    - Al finalizar la iteración:
      - Demostrar al cliente los requisitos completados en cada iteración.
      - Hacer una retrospectiva al final de cada iteración para mejorar de forma continua su manera de trabajar.
  - El equipo es multidisciplinar:
    - Los miembros del equipo tienen las habilidades necesarias para poder identificar y ejecutar todas las tareas que permiten proporcionar al cliente los requisitos comprometidos en la iteración.
    - Tienen que depender lo mínimo de personas externas al equipo, de manera que el compromiso que adquieren en cada iteración no se ponga en peligro.
    - Se crea una sinergia que permite que el resultado sea más rico al nutrirse de las diferentes experiencias, conocimientos y habilidades de todos. Colaboración creativa.
  - Los miembros del equipo deben dedicarse al proyecto a tiempo completo para evitar dañar su productividad por cambios de tareas en diferentes proyectos, para evitar interrupciones externas y así poder mantener el compromiso que adquieren en cada iteración.

- Todos los miembros del equipo trabajan en la misma localización física, para poder maximizar la comunicación entre ellos mediante conversaciones cara a cara, diagramas en pizarras blancas, etc. De esta manera se minimizan otros canales de comunicación menos eficientes, que hacen que las tareas se transformen en un "pasa pelota" o que hacen perder el tiempo en el establecimiento de la comunicación (como cuando se llama repetidas veces por teléfono cuando la persona no está en su puesto).
- El equipo debe ser estable durante el proyecto, sus miembros deben cambiar lo mínimo posible, para poder aprovechar el esfuerzo que les ha costado construir sus relaciones interpersonales, engranarse y establecer su organización del trabajo.

3.4.1.2. Roles "Gallina"

Los roles gallina en realidad no son parte del proceso Scrum, pero deben tenerse en cuenta. Un aspecto importante de una aproximación ágil es la práctica de involucrar en el proceso a los usuarios, expertos del negocio y otros interesados (stakeholders). Es importante que estas personas participen y entreguen retroalimentación con respecto a la salida del proceso a fin de revisar y planear de cada sprint. Entre estos roles se encuentran:

- Usuarios: Aquellos que utilizaran el software.
- Stakeholders (Clientes, Proveedores): Se refiere a las personas que hacen posible el proyecto y para quienes el proyecto producirá el beneficio acordado que lo justifica. Sólo participan directamente durante las revisiones del sprint.
- Managers: Son las personas que establecen el ambiente para el desarrollo del producto.

3.4.2. Elementos y documentos de Scrum

- **Product backlog**: El product backlog es un documento de alto nivel para todo el proyecto. Contiene descripciones genéricas de todos los requerimientos, funcionalidades deseables, riesgos, etc. priorizadas según su valor para el negocio (Business value). Es el qué va a ser construido. Es abierto y cualquiera puede modificarlo. Contiene estimaciones tanto del valor para el negocio, como del esfuerzo de desarrollo requerido. Esta estimación ayuda al product Owner a ajustar la línea temporal y, de manera limitada, la prioridad de las diferentes tareas. En la figura 3.11. puede verse un ejemplo del Product Backlog.

Área de requisitos	Requisitos	Origen	Valor	Estimación inicial	Factor Ajuste	Estimación ajustada	Iteración: 1	2	3	4	5
							Pendiente: 225	170	114	57	
Área X	Requisito A	Marketing	2000	15		15	15	0			
Área Z	Requisito B	Producción	1750	20		20	20	0			
Área Y	Requisito C	Ventas	1500	20		20	20	0			
	Iteración 1		5250	55		55	55	0	0	0	0
Área Z	Requisito C	Producción	1250	15	0,2	18	18	18	0		
Área X	Requisito D	Producción	1250	20		20	20	20	0		
Área Z	Requisito E	Marketing	1000	15	0,2	18	18	18	0		
	Iteración 2		3500	50		56	56	56	0	0	0
	Primera entrega		8750	105		111	111	56	0	0	0
Área X	Requisito F	Marketing	1250	20	0,2	24	24	24	24	0	
Área Y	Requisito G	Marketing	750	15		15	15	15	15	0	
Área Y	Requisito H	Ventas	750	15	0,2	18	18	18	18	0	
	Iteración 3		2750	50		57	57	57	57	0	0
Área Z	Requisito I	Producción	700	15	0,2	18	18	18	18	18	
Área Y	Requisito J	Marketing	500	10	0,5	15	15	15	15	15	
Área Y	Requisito K	Ventas	500	20	0,2	24	24	24	24	24	
	Iteración 4		1700	45		57	57	57	57	57	0
	Segunda entrega		4450	95		114	114	114	114	57	0

Figura 3.11. Ejemplo Product backlog

- **Sprint Backlog**: Lista de los trabajos que realizará el equipo durante el sprint para generar el incremento previsto. El equipo asume el compromiso de la ejecución. Las

tareas están asignadas a personas, y tienen estimados el tiempo y los recursos necesarios. Las tareas se dividen en *horas* con ninguna tarea de duración superior a cierta cantidad máxima de horas (se recomienda aproximadamente 16 horas). Si una tarea es mayor que el máximo de horas, deberá ser dividida en mayor detalle. Las tareas en el *sprint backlog* nunca son asignadas, son tomadas por los miembros del equipo del modo que les parezca oportuno. En la figura 3.12. puede verse un ejemplo del Sprint Backlog.

Requisito	Tarea	Quien	Estado (No iniciada / en progreso / completada)	Dia:	1	2	3	4	5	6	7	8	9	10
				Horas pendientes	1120	1088	1076	1048	1040	1032	1020	1008	992	972
Requisito A	Tarea 1	Joao	Completada		16	8								
Requisito A	Tarea 4	Laura	Completada		4									
Requisito A	Tarea 5	Laura	Completada		4									
Requisito A	Tarea 3	Gabri	Completada		8									
Requisito A	Tarea 2	Laura	Completada		16	8	4							
Requisito A	Tarea 6	Gabri	Completada		8	8	8							
Requisito A	Tarea 7	Joao	Completada		16	16	16	8						
Requisito A	Tarea 8	Laura	Completada		8	8	8							
Requisito A	Tarea 9	Laura	Completada		8	8	8	8	8					
Requisito A	Tarea 10	Laura	Completada		8	8	8	8	8	8	4			
Requisito A	Tarea 11	Joao	Completada		16	16	16	16	16	16	8			
Requisito B	Tarea 12	Gabri	Completada		16	16	16	16	16	16	16	16	8	
Requisito B	Tarea 13	Laura	Completada		16	16	16	16	16	16	16	16	8	
Requisito B	Tarea 14	Joao	En progreso		8	8	8	8	8	8	8	8	8	4
Requisito B	Tarea 15	Gabri	En progreso		8	8	8	8	8	8	8	8	8	8
Requisito B	Tarea 16	Laura	En progreso		8	8	8	8	8	8	8	8	8	8
Requisito C	Tarea 17	Joao	No iniciada		4	4	4	4	4	4	4	4	4	4
Requisito C	Tarea 18	Gabri	No iniciada		8	8	8	8	8	8	8	8	8	8
Requisito C	Tarea 19	Laura	No iniciada		16	16	16	16	16	16	16	16	16	16
Requisito C	Tarea 20	Joao	No iniciada		8	8	8	8	8	8	8	8	8	8

Figura 3.12. Ejemplo Sprint backlog

- **Burn Down Chart**: La *burn down chart* es una gráfica mostrada públicamente que mide la cantidad de requisitos en el Backlog del proyecto pendientes al comienzo de cada Sprint. Dibujando una línea que conecte los puntos de todos los Sprints completados, podremos ver el progreso del proyecto. Lo normal es que esta línea sea descendente (en casos en que todo va bien en el sentido de que los requisitos están bien definidos desde el principio y no varían nunca) hasta llegar al eje horizontal, momento en el cual el proyecto se ha terminado (no hay más requisitos pendientes de ser completados en el Backlog). Si durante el proceso se añaden nuevos requisitos la recta tendrá pendiente ascendente en determinados segmentos, y si se modifican algunos requisitos la pendiente variará o incluso valdrá cero en algunos tramos. En la figura 3.13. se muestra un ejemplo de este diagrama:

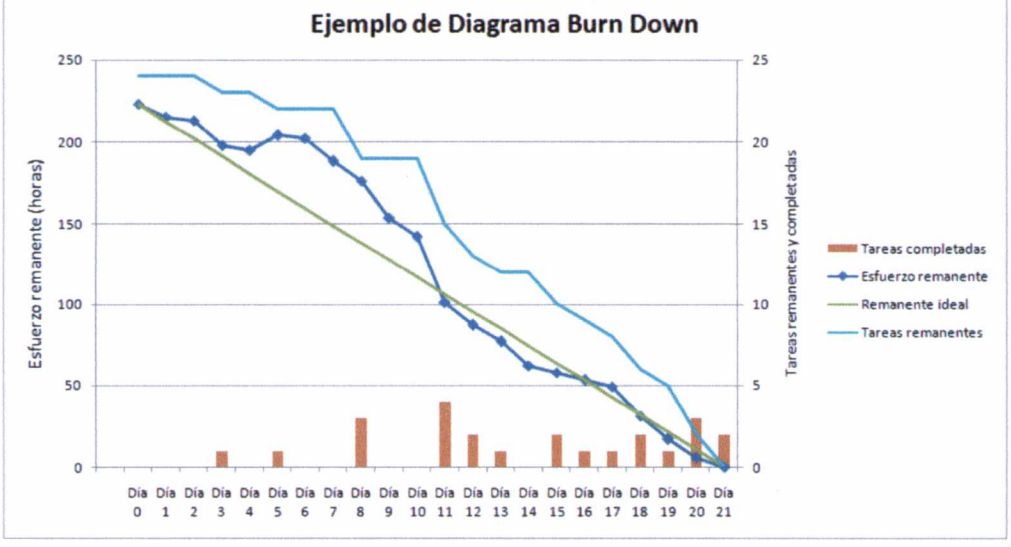

Figura 3.13. Ejemplo de Burn Down Chart

- **Burn Up Chart**: Herramienta de gestión y seguimiento para el propietario del producto. Presenta de un vistazo las versiones de producto previstas, las funcionalidades de cada una, velocidad estimada, fechas probables para cada versión, margen de error previsto en las estimaciones, y avance real. En la figura 3.14. se da un ejemplo de este diagrama.

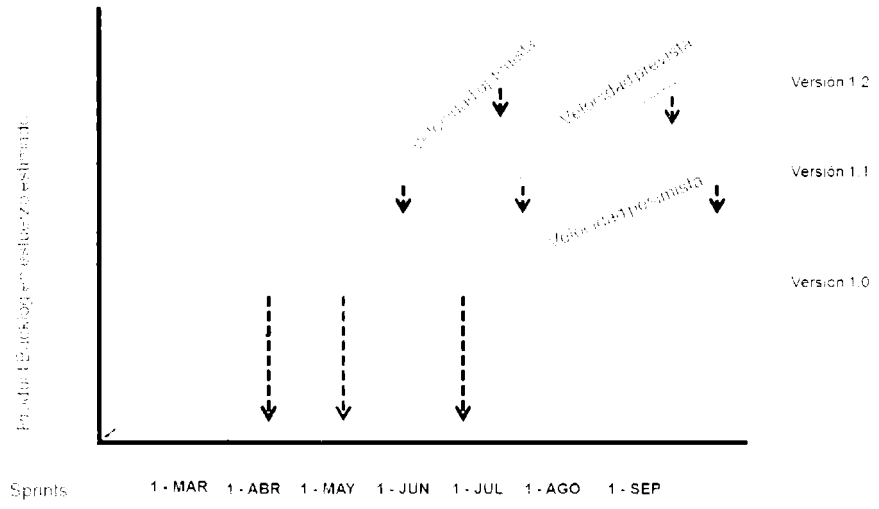

Figura 3.14. Ejemplo de Burn Up Chart

- **Incremento**: Resultado de cada sprint. Se trata de un resultado completamente terminado y en condiciones de ser usado.

3.4.3. Proceso Scrum

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas, si así se necesita). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente prioriza los objetivos balanceando el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas. De manera regular el cliente puede maximizar la utilidad de lo que se desarrolla y el retorno de inversión mediante la replanificación de objetivos que realiza al inicio de cada iteración.

Las actividades que se llevan a cabo en Scrum son las siguientes:

- **Planificación de la iteración (Sprint Planning):** Tiene dos partes:
  - **Selección de requisitos**
  - **Planificación de la iteración**
- **Ejecución de la iteración (Sprint)**
- **Reunión diaria de sincronización del equipo (Scrum daily meeting)**
- **Demostración de requisitos completados (Sprint Demostration)**
- **Retrospectiva (Sprint Retrospective)**
- **Replanificación del proyecto**

En la figura 3.15. se muestra un grafico sintético del proceso Scrum.

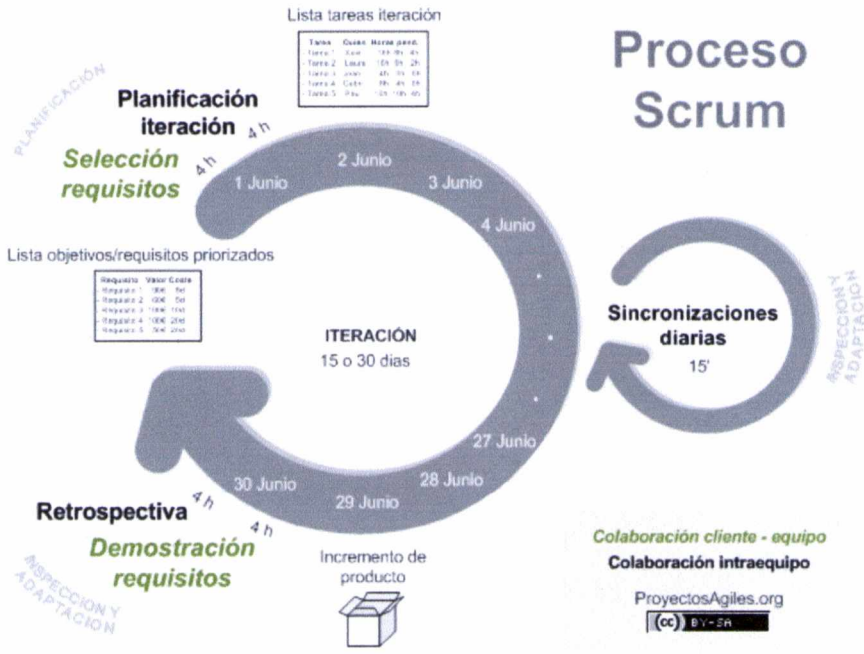

Figura 3.15. Proceso Scrum

En los próximos apartados se explicaran con más detalle las respectivas actividades.

3.4.3.1. Planificación de la iteración (Sprint Planning)

La planificación de las tareas a realizar en la iteración se divide en dos partes:

- Primera parte de la reunión. Se realiza en un timebox de cómo máximo 4 horas:
  - El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto, pone nombre a la meta de la iteración (de manera que ayude a tomar decisiones durante su ejecución) y propone los requisitos más prioritarios a desarrollar en ella.
  - El equipo examina la lista, pregunta al cliente las dudas que le surgen, añade más condiciones de completitud y selecciona los objetivos/requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.
- Segunda parte de la reunión. Se realiza en un timebox de cómo máximo 4 horas. El equipo planifica la iteración, elabora la táctica que le permitirá conseguir el mejor resultado posible con el mínimo esfuerzo. Esta actividad la realiza el equipo dado que ha adquirido un compromiso, es el responsable de organizar su trabajo y es quien mejor conoce cómo realizarlo.
  - Se definen las tareas necesarias para poder completar cada objetivo/requisito, creando la lista de tareas de la iteración.
  - Se realizan estimaciones conjuntas del esfuerzo necesario para realizar cada tarea.
  - Cada miembro del equipo se autoasigna a las tareas que puede realizar.

Beneficios

- Productividad mediante comunicación y creación de sinergias:
  - Todos los miembros del equipo tienen una misma visión del objetivo y se ha utilizado los conocimientos y las experiencias de todos para elaborar la mejor solución entregable en el mínimo tiempo y con el mínimo esfuerzo, eliminando tareas innecesarias, detectando conflictos y dependencias entre tareas, etc.
- Potenciación del compromiso del equipo:
  - Es el equipo quien asume la responsabilidad de completar en la iteración los requisitos que selecciona.
  - Es cada una de las personas la que se responsabiliza de realizar las tareas a las que se autoasigna.
- Una estimación conjunta es más fiable, dado que tiene en cuenta los diferentes conocimientos, experiencia y habilidades de los integrantes del equipo.

3.4.3.2. Ejecución de la iteración (Sprint)

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos. Cada iteración tiene que proporcionar un resultado completo, un incremento de producto que sea susceptible de ser entregado con el mínimo esfuerzo cuando el Product Owner lo solicite.

Cada día el equipo realiza una reunión de sincronización, donde cada miembro inspecciona el trabajo de los otros para poder hacer las adaptaciones necesarias, comunica cuales son los impedimentos con que se encuentra, actualiza el estado de la lista de tareas de la iteración (Sprint Backlog) y los gráficos de trabajo pendiente (Burndown charts).

El Scrum Master se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad, eliminando los obstáculos que el equipo no puede resolver por sí mismo y protegiendo al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

Sólo en situaciones muy excepcionales el Product Owner o el equipo pueden solicitar una terminación anormal de la iteración. Esto puede suceder si, por ejemplo, el contexto del proyecto ha cambiado enormemente y no es posible esperar al final de la iteración para aplicar cambios, o si el equipo encuentra que es imposible cumplir con el compromiso adquirido. En ese caso, se dará por finalizada la iteración y se dará inicio a otra mediante una reunión de planificación de la iteración.

Recomendaciones

- Para poder completar el máximo de requisitos en la iteración, se debe minimizar el número de objetivos/requisitos en que el equipo trabaja simultáneamente (*WIP, Work In Progress*), completando primero los que den más valor al cliente. Esta forma de trabajar, que se ve facilitada por la propia estructura de la lista de tareas de la iteración, permite tener más capacidad de reacción frente a cambios o situaciones inesperadas.

Restricciones

- No se puede cambiar los objetivos/requisitos de la iteración en curso.

3.4.3.3. Reunión diaria de sincronización del equipo (Scrum daily meeting)

El objetivo de esta reunión es facilitar la transferencia de información y la colaboración entre los miembros del equipo para aumentar su productividad, al poner de manifiesto puntos en que se pueden ayudar unos a otros.

Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para, al finalizar la reunión, poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso conjunto que el equipo adquirió para la iteración (en la reunión de planificación de la iteración).

Cada miembro del equipo debe responder las siguientes preguntas en un timebox de cómo máximo 15 minutos:

- ¿Qué he hecho desde la última reunión de sincronización? ¿Pude hacer todo lo que tenía planeado? ¿Cuál fue el problema?
- ¿Qué voy a hacer a partir de este momento?
- ¿Qué impedimentos tengo o voy a tener para cumplir mis compromisos en esta iteración y en el proyecto?

Como apoyo a la reunión, el equipo cuenta con la lista de tareas de la iteración, donde se actualiza el estado y el esfuerzo pendiente para cada tarea, así como con el gráfico de horas pendientes en la iteración.

Beneficios

- Aumentar la productividad en el proyecto y potenciar el compromiso de equipo.
- Fomentar el aprendizaje de los miembros del equipo, ya que pueden ver cómo trabajan los otros según sus especialidades y experiencias.
- Conocer el estado de la iteración, ver si es posible completar los requisitos a que se comprometió el equipo, en vista de las desviaciones y de las tareas pendientes.

Restricciones

- La reunión diaria de estado y sincronización del equipo no es para resolver problemas, los problemas se resuelven después de la reunión.
- El equipo debe contar con unos criterios consensuados sobre el proceso de ejecución de las de tareas.

Recomendaciones

- Realizar la reunión diaria de sincronización de pie, para que los miembros del equipo no se relajen ni se extiendan en más detalles de los necesarios.
- Realizar las reuniones de colaboración entre miembros del equipo justo después de la de sincronización.

3.4.3.4. Demostración de requisitos completados (Sprint Demonstration)

Reunión informal donde el equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo, haciendo un recorrido por ellos lo más real y cercano posible al objetivo que se pretende cubrir.

En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.

Se realiza en un timebox de cómo máximo 4 horas.

Beneficios

- El cliente puede ver de manera objetiva cómo han sido desarrollados los requisitos que proporcionó, ver si se cumplen sus expectativas, entender más qué es lo que necesita y tomar mejores decisiones respecto al proyecto.
- El equipo puede ver si realmente entendió cuáles eran los requisitos que solicitó el cliente y ver en qué puntos hay que mejorar la comunicación entre ambos.
- El equipo se siente más satisfecho cuando puede ir mostrando los resultados que va obteniendo. No está meses trabajando sin poder exhibir su obra.

3.4.3.5. Retrospectiva (Sprint Retrospective)

Con el objetivo de mejorar de manera continua su productividad, el equipo analiza cómo ha sido su manera de trabajar durante la iteración. Para esto, se analiza por participante:

- Qué cosas han funcionado bien?
- Cuales hay que mejorar?
- Qué cosas quiere probar hacer en la siguiente iteración?
- Qué ha aprendido?
- Cuales son los problemas que podrían impedirle progresar adecuadamente? El Scrum Master se encargará de ir eliminando los obstáculos identificados que el propio equipo no pueda resolver por sí mismo.

Esta reunión se realiza después de la reunión de demostración al cliente de los objetivos conseguidos en la iteración, para poder incorporar su feedback y cumplimiento de expectativas como parte de los temas a tratar en la reunión de retrospectiva

Se realiza en un timebox de cómo máximo 3 horas.

Beneficios

- Incrementa la productividad en el proyecto y el aprendizaje del equipo de manera sistemática, iteración a iteración, con resultados a corto plazo.
- Aumenta la motivación del equipo dado que participa en la mejora de proceso, se siente escuchado, toma decisiones consensuadas (y más sostenibles) para ir eliminando lo que le molesta y le impide que sea más productivo.

Restricciones

- Es necesario que el Equipo y el Facilitador dispongan de autoridad, mecanismos y recursos para ir mejorando su forma de trabajar y el contexto del proyecto. Es frustrante encontrar sistemáticamente los mismos obstáculos y no poder solucionarlos.

3.4.3.6. Replanificación del proyecto

En las reuniones de planificación de entregas y/o durante el transcurso de una iteración, el cliente va trabajando en la lista de objetivos/requisitos priorizada del producto o proyecto, añadiendo requisitos, modificándolos, eliminándolos, repriorizándolos, cambiando el contenido de iteraciones y definiendo un calendario de entregas que se ajuste mejor a sus nuevas necesidades.

Los cambios en la lista de requisitos pueden ser debidos a:

- Modificaciones que el cliente solicita tras la demostración que el equipo realiza al final de cada iteración sobre los resultados obtenidos, ahora que el cliente entiende mejor el producto o proyecto.
- Cambios en el contexto del proyecto (sacar al mercado un producto antes que su competidor, hacer frente a urgencias o nuevas peticiones de clientes, etc.).
- Nuevos requisitos o tareas como resultado de nuevos riesgos en el proyecto.
- Etc.

Para realizar esta tarea, el cliente colabora con el equipo:

- Para cada nuevo objetivo/requisito conjuntamente se hace una identificación inicial de sus condiciones de completitud (que se detallarán en la reunión de planificación de la iteración).
- El Product Owner obtiene del equipo la re-estimación de costes de desarrollo para completar los objetivos/requisitos siguientes. El equipo ajusta el factor de complejidad, el coste para completar los requisitos y su velocidad de desarrollo en función de la experiencia adquirida hasta ese momento en el proyecto.
- El Product Owner re-prioriza la lista de objetivos/requisitos en función de estas nuevas estimaciones.

Hay que notar que el equipo sigue trabajando con los objetivos/requisitos de la iteración en curso, (que de hecho eran los más prioritarios al iniciar la iteración). No es posible cambiar los requisitos que se desarrollan durante la iteración. En la reunión de planificación de la iteración el Product Owner presentará la nueva lista de requisitos para que sea desarrollada.

Beneficios

De manera sistemática, iteración a iteración, se obtienen los siguientes beneficios:

- El cliente puede tomar decisiones con tiempo respecto al progreso del proyecto y posibles desviaciones.
- El plan de proyecto se actualiza con la velocidad de desarrollo del equipo, se evitan sorpresas de última hora.

3.4.4. Controles de Scrum

La metodología Scrum implementa controles haciendo uso de técnicas orientadas a objetos.

El riesgo es el control primario. Al asumir el riesgo se provocan cambios en otros controles. Los controles en la metodología Scrum son:

- **Backlog:** Los requisitos y funcionalidades del producto que no están recogidos en el entregable actual. Bugs, defectos, peticiones de mejora del cliente, funcionalidades del producto competitivas, actualizaciones de la tecnología empleada son todos ítems del backlog.
- **Entregable/mejora:** Son los ítems del backlog que en un momento determinado representan un posible entregable teniendo en cuenta las variables de requisitos, tiempo, calidad, etc.
- **Paquetes:** Se trata de los componentes u objetos que deben ser modificados para poder implementar los ítems del backlog.
- **Cambios:** Los cambios a realizar a un paquete para poder implementar a un ítem del backlog.
- **Problemas:** El conjunto de problemas técnicos que ocurren y deben ser resueltos para poder implementar un cambio.
- **Riesgos:** Los riesgos que amenazan el éxito del proyecto se deben identificar y planificar respuestas a los mismos.
- **Soluciones:** Soluciones a los problemas y riesgos identificados, algunos resultan en cambios.

3.4.5. Proceso completo de Scrum

En la Figura 3.16 se muestra a modo grafico el proceso completo de Scrum.

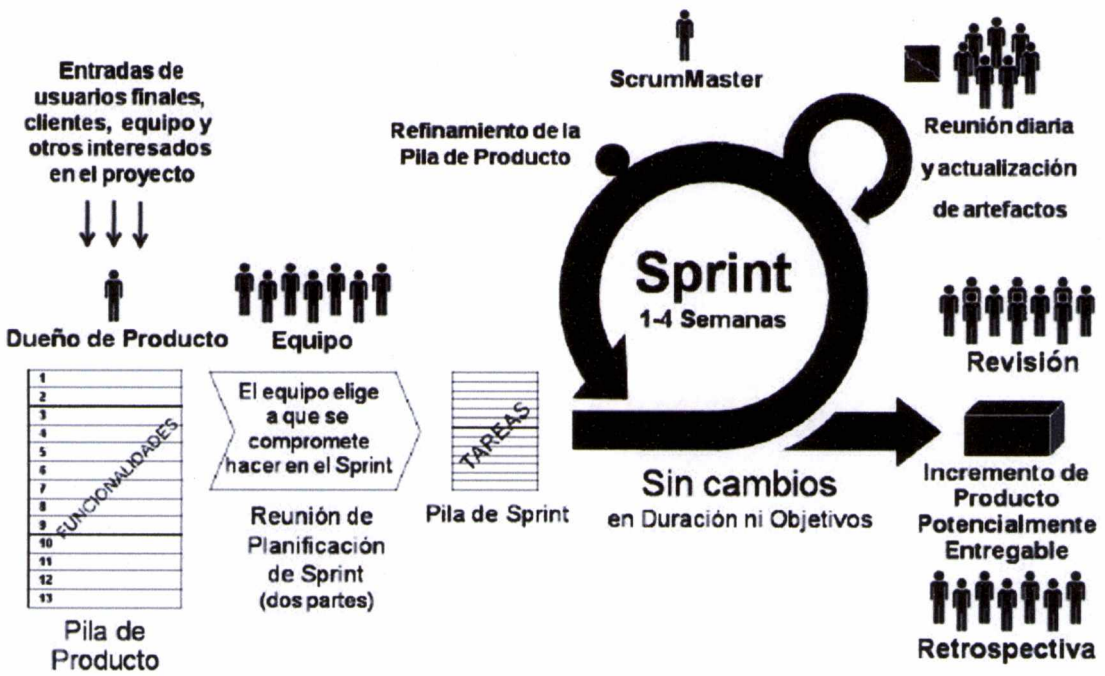

Figura 3.16. Proceso Scrum completo

Capítulo 4

Herramientas útiles para el desarrollo de Software

RTC se presenta como una herramienta eficaz para el desarrollo de software. Pero obviamente no es la única. Este capitulo presenta algunas herramientas alternativas para ese fin, indicando propiedades como especificaciones técnicas, arquitectura, características, ventajas y desventajas (según la herramienta se pudo recopilar determinada información). Se explican algunos software como Subversión, Git, Mercurial, SourceSafe y Team Foundation Server (Versiones de Microsoft, competidor natural de IBM), entre otros.

Este apartado tiene como objetivo presentar otras herramientas aptas para el desarrollo de software y describir sus especificaciones y características.

4.1. CVS

Concurrent Versions System (CVS), también conocido como Concurrent Versioning System, es una aplicación que implementa un sistema de control de versiones. Mantiene el registro de todo el trabajo y los cambios en los ficheros (código fuente principalmente) que forman un proyecto (de programa) y permite que distintos desarrolladores (potencialmente situados a gran distancia) colaboren. CVS se ha hecho popular en el mundo del software libre. Sus desarrolladores difunden el sistema bajo la licencia GPL.

4.1.1. Características

CVS utiliza una arquitectura cliente-servidor: un servidor guarda la(s) versión(es) actual(es) del proyecto y su historial. Los clientes se conectan al servidor para sacar una copia completa del proyecto. Esto se hace para que eventualmente puedan trabajar con esa copia y más tarde ingresar sus cambios con comandos GNU.

Típicamente, cliente y servidor se conectan utilizando Internet, pero con el sistema CVS el cliente y servidor pueden estar en la misma máquina. El sistema CVS tiene la tarea de mantener el registro de la historia de las versiones del programa de un proyecto solamente con desarrolladores locales. Originalmente, el servidor utilizaba un sistema operativo similar a Unix, aunque en la actualidad existen versiones de CVS en otros sistemas operativos, incluido Windows. Los clientes CVS pueden funcionar en cualquiera de los sistemas operativos más difundidos.

Varios clientes pueden sacar copias del proyecto al mismo tiempo. Posteriormente, cuando actualizan sus modificaciones, el servidor trata de acoplar las diferentes versiones. Si esto falla, por ejemplo debido a que dos clientes tratan de cambiar la misma línea en un archivo en particular, entonces el servidor deniega la segunda actualización e informa al cliente sobre el conflicto, que el usuario deberá resolver manualmente. Si la operación de ingreso tiene éxito, entonces los números de versión de todos los archivos implicados se incrementan automáticamente, y el servidor CVS almacena información sobre la actualización, que incluye una descripción suministrada por el usuario, la fecha y el nombre del autor y sus archivos log.

Los clientes pueden también comparar diferentes versiones de archivos, solicitar una historia completa de los cambios, o sacar una "foto" histórica del proyecto tal como se encontraba en una fecha determinada o en un número de revisión determinado.

Muchos proyectos de código abierto permiten el "acceso de lectura anónimo", significando que los clientes pueden sacar y comparar versiones sin necesidad de teclear una contraseña; solamente el ingreso de cambios requiere una contraseña en estos casos.

Los clientes también pueden utilizar el orden de actualización con el fin de tener sus copias al día con la última versión que se encuentra en el servidor. Esto elimina la necesidad de repetir las descargas del proyecto completo.

CVS también puede mantener distintas "ramas" de un proyecto. Por ejemplo, una versión difundida de un proyecto de programa puede formar una rama y ser utilizada para corregir errores. Todo esto se puede llevar a cabo mientras la versión que se encuentra actualmente en desarrollo y posee cambios mayores con nuevas características se encuentre en otra línea formando otra rama separada.

4.1.2. Ventajas

- *Software bajo licencia GPL.*
- *Permite la concurrencia del trabajo*, donde más de una persona trabaja con el mismo archivo.
- *Variedad de clientes a escoger.*
- *Manejo de múltiples versiones simultaneas.*
- *Integración de versiones.*
- *Algoritmos de comparación de contenido.*

4.1.3. Desventajas

- *No soporta refactorización de sistemas de forma automática o versionada.*
- *Limitado para UTF-8, Unicode o archivos con contenido diferente a ASCII.*
- *El protocolo no soporta eliminación ni renombre de directorios o archivos.*
- *Depende de la implementación de los clientes.*
- *Varias plataformas carecen de soporte nativo a CVS*
- *Los COMMIT no son atómicos*
- *CVS trata con archivos de texto por defecto*

- *No hay soporte para control de versiones distribuidos o cambios no publicados.* Los programadores deben confirmar los cambios de los archivos a menudo por la fusión y frecuente publicación.

4.2. Subversion

Subversion es un sistema de control de versiones libre y de código fuente abierto. Maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único (si se ha hecho un cambio incorrecto a los datos, simplemente debe deshacerse ese cambio).

> Algunos sistemas de control de versiones son también sistemas de administración de configuración de software. Estos sistemas son diseñados específicamente para la administración de árboles de código fuente, y tienen muchas características que son específicas del desarrollo de software (tales como el entendimiento nativo de lenguajes de programación, o el suministro de herramientas para la construcción de software). Sin embargo, Subversion no es uno de estos sistemas.

Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros. Puede tratarse de ficheros de código fuente pero también de cualquier otra cosa (como una lista de compra de comestibles).

4.2.1. Características

Para discutir las características que Subversion aporta al control de versiones, se expondrá como ha evolucionado respecto al diseño de CVS (su antecesor).

Subversion proporciona:

- *Versionado de directorios:* CVS solamente lleva el historial de ficheros individuales, pero Subversion implementa un sistema de ficheros versionado "virtual" que sigue los cambios sobre árboles de directorios completos a través del tiempo. Ambos, ficheros y directorios, se encuentran bajo el control de versiones.
- *Verdadero historial de versiones:* Dado que CVS está limitado al versionado de ficheros, operaciones como copiar y renombrar (las cuales pueden ocurrir sobre ficheros, pero que realmente son cambios al contenido del directorio en el que se encuentran) no son soportadas por CVS. Adicionalmente, en CVS no se puede reemplazar un fichero versionado con un fichero nuevo que lleve el mismo nombre sin que el nuevo elemento herede el historial del fichero antiguo (que quizás sea completamente distinto al anterior). Con Subversion, se pueden añadir, borrar, copiar, y renombrar ficheros y directorios. Y cada fichero nuevo añadido comienza con un historial nuevo, limpio y completo.
- *Envíos atómicos:* Una colección cualquiera de modificaciones o bien entra por completo al repositorio, o bien no lo hace en absoluto. Esto permite a los desarrolladores construir y enviar los cambios como fragmentos lógicos e impide que ocurran problemas cuando sólo una parte de los cambios enviados lo hace con éxito.
- *Versionado de metadatos:* Cada fichero y directorio tiene un conjunto de propiedades (claves y sus valores) asociado a él. Se pueden crear y almacenar cualquier par arbitrario de clave/valor que se desee. Las propiedades son versionadas a través del tiempo, al igual que el contenido de los ficheros.

- *Elección de las capas de red:* Subversion tiene una noción abstracta del acceso al repositorio, facilitando a las personas implementar nuevos mecanismos de red. Subversion puede conectarse al servidor HTTP Apache como un módulo de extensión. Esto proporciona a Subversion una gran ventaja en estabilidad e interoperabilidad, y acceso instantáneo a las características existentes que ofrece este servidor (autenticación, autorización, compresión de la conexión, etcétera). También tiene disponible un servidor de Subversion independiente, y más ligero. Este servidor habla un protocolo propio, el cual puede ser encaminado fácilmente a través de SSH.
- *Manipulación consistente de datos:* Subversion expresa las diferencias del fichero usando un algoritmo de diferenciación binario, que funciona idénticamente con ficheros de texto y ficheros binarios. Ambos tipos de ficheros son almacenados igualmente comprimidos en el repositorio, y las diferencias son transmitidas en ambas direcciones a través de la red.
- *Ramificación y etiquetado eficientes:* El coste de ramificación y etiquetado no necesita ser proporcional al tamaño del proyecto. Subversion crea ramas y etiquetas simplemente copiando el proyecto. De este modo estas operaciones toman solamente una cantidad de tiempo pequeña y constante.
- *Hackability:* Subversion está implementado como una colección de bibliotecas compartidas en C con APIs bien definidas. Esto hace a Subversion extremadamente fácil de mantener y reutilizable por otras aplicaciones y lenguajes.

4.2.2. Arquitectura de Subversion

En un extremo se encuentra un repositorio de Subversion que conserva todos los datos versionados. Al otro lado, hay un programa cliente Subversion que administra réplicas parciales de esos datos versionados (llamadas "copias de trabajo"). Entre estos extremos hay múltiples rutas a través de varias capas de acceso al repositorio (AR). Algunas de estas rutas incluyen redes de ordenadores y servidores de red que después acceden al repositorio. Otras pasan por alto la red y acceden al repositorio directamente.

Componentes de Subversion
Una vez instalado, Subversion se compone de un número diferente de piezas. A continuación se presenta una visión general de estos componentes.

- Svn: El programa cliente de línea de comandos.
- Svnversion: Programa para informar del estado (en términos de revisiones de los elementos presentes) de una copia de trabajo.
- Svnlook: Una herramienta para inspeccionar un repositorio de Subversion.
- Svnadmin: Herramienta para crear, modificar o reparar un repositorio de Subversion.
- Svndumpfilter: Un programa para filtrar el formato de salida de volcado de repositorios Subversion.
- mod_dav_svn: Un módulo para el servidor HTTP Apache usado para hacer que su repositorio esté disponible a otros a través de una red.
- Svnserve: Un servidor independiente, ejecutable como proceso demonio o invocable por SSH; otra manera de hacer que su repositorio esté disponible para otros a través de una red.

En la figura 4.1. se muestra gráficamente la arquitectura de Subversion.

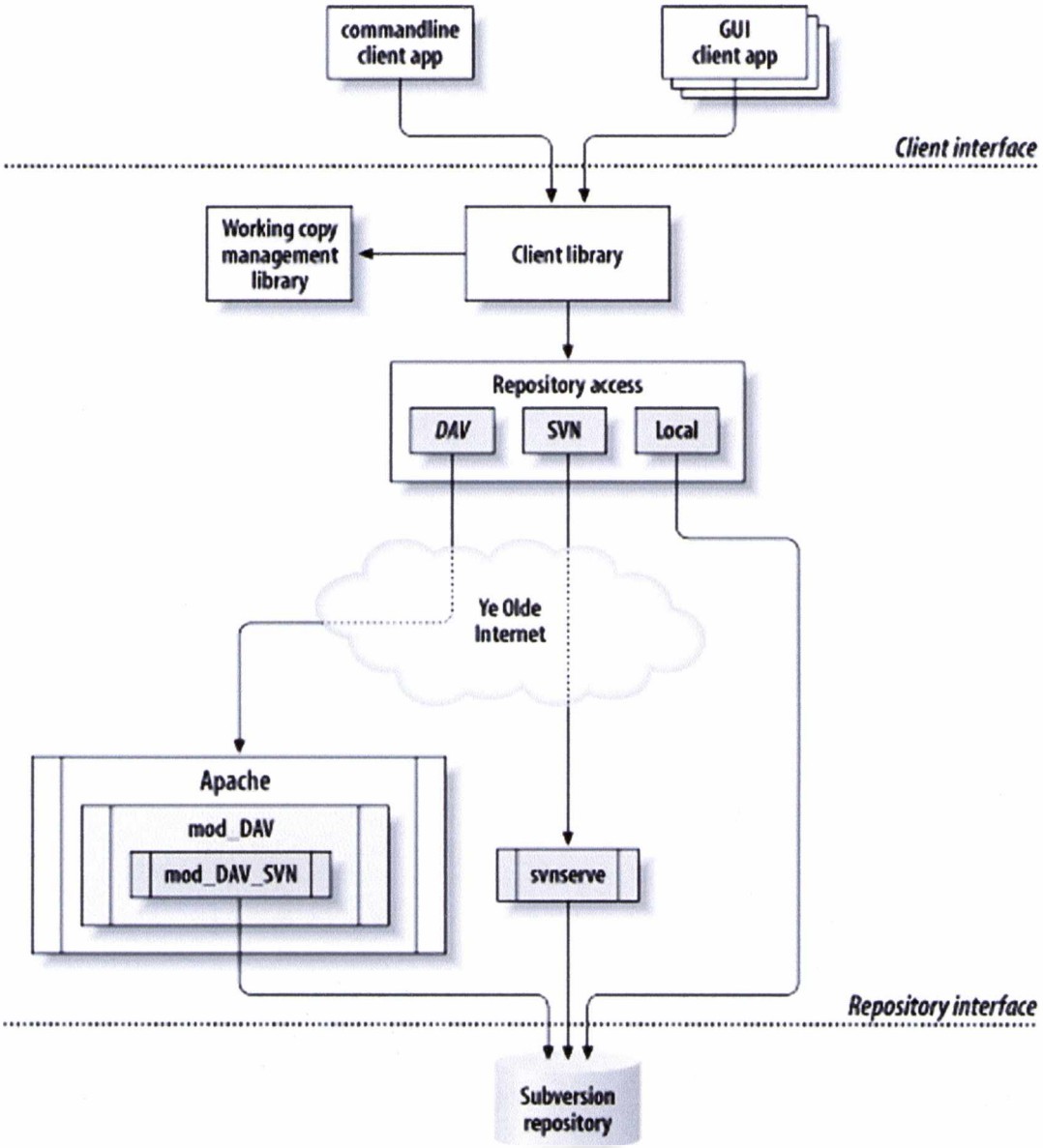

Figura 4.1. Arquitectura de Subversion

4.2.3. Ventajas

- *Se sigue la historia de los archivos y directorios a través de copias y renombrados.*
- Las modificaciones (incluyendo cambios a varios archivos) son *atómicas*.
- La creación de ramas y etiquetas es una operación más eficiente; Tiene costo de *complejidad constante* (O(1)) y no lineal (O(n)) como en CVS.
- *Se envían sólo las diferencias en ambas direcciones* (en CVS siempre se envían al servidor archivos completos).
- Puede ser servido mediante Apache, sobre WebDAV/DeltaV. Esto permite que clientes WebDAV utilicen Subversion en forma *transparente*.
- *Maneja eficientemente archivos binarios* (a diferencia de CVS que los trata internamente como si fueran de texto).
- *Permite selectivamente el bloqueo de archivos.* Se usa en archivos binarios que, al no poder fusionarse fácilmente, conviene que no sean editados por más de una persona a la vez.

- *Cuando se usa integrado a Apache permite utilizar todas las opciones que este servidor provee a la hora de autentificar archivos (SQL, LDAP, PAM, etc.).*

4.2.4. Desventajas

- *El manejo de cambio de nombres de archivos no es completo*. Lo maneja como la suma de una operación de copia y una de borrado.
- No resuelve el problema de aplicar repetidamente parches entre ramas, no facilita el llevar la cuenta de qué cambios se han trasladado. Esto se resuelve siendo cuidadoso con los mensajes de COMMIT. Esta carencia es parcialmente corregida en la versión 1.5.

4.3. SourceSafe

Microsoft Visual SourceSafe (también conocido por sus siglas VSS) es una herramienta de Control de versiones que forma parte de Microsoft Visual Studio aunque está siendo sustituida por el Visual Studio Team Foundation Server.

4.3.1. Ventajas

Para las personas que desarrollan programas en el sistema operativo Windows, resulta una herramienta útil ya que se integra fuertemente con el entorno de desarrollo integrado o IDE de Visual Studio permitiendo un manejo relativamente simple de versiones sobre una computadora individual y en equipos de trabajo relativamente pequeños.

4.3.2. Desventajas

La principal desventaja de Visual SourceSafe reside en el método de acceso a los archivos compartidos que constituyen su repositorio mediante el protocolo SMB que no impide que éstos sean manipulados de manera externa al producto por cualquier persona que tenga acceso al mismo, provocando corrupción de datos. Este mismo tipo de acceso a archivos compartidos provoca que en equipos de trabajo grandes, el acceso concurrente pueda ser particularmente lento.

SourceSafe es configurable, permitiendo que un solo programador modifique el código fuente (recomendado) o que lo hagan varios. Las herramientas de gestión de diferencias para reunificar el código fuente modificado por varios programadores no son demasiado buenas comparadas con las de otros gestores de código fuente.
Además, SourceSafe es inestable cuando se suben ficheros binarios de gran tamaño, ya que espera solo ficheros de texto. Así que no es apto para almacenar documentación, sólo código fuente.

Otras desventajas y limitaciones importantes:

- *SourceSafe no tiene soporte para revisiones alternas o de prueba (branching):* Un sistema de control de revisiones debe proveer soporte para *branching* (revisiones alternas). Con un sistema adecuado de *branching* los desarrolladores pueden hacer revisiones menores de versiones anteriores o actuales mientras se sigue el desarrollo con la versión principal para una nueva versión y pueden probar actualizaciones a nivel proyecto sin afectar la versión funcional. De igual manera debe tener un sistema de diferenciación entre "*branches*" o versiones para poder modificarlos, unirlos, o deshacer cambios. El "*merge*" de SourceSafe está integrado ampliamente con el *check-in*, haciendo que sea difícil validar las diferencias de una propuesta de merge antes de mandarlo a *check-in*.
- *No se puede extender a SourceSafe de manera segura:* Debería ser posible añadirle funcionalidades al sistema de control de revisiones. Por ejemplo, envío de correos haciendo un resumen de check-ins es muy importante. Al trabajar con un equipo, emails con mensajes listando archivos actualizados así como las actividades realizadas en cada uno, pueden mantener a todo el equipo al tanto de las actualizaciones. Quizá también se requiera agregar filtros para prevenir que se actualicen archivos que no cumplen con ciertos requerimientos. SourceSafe apenas soporta algo remotamente parecido. Aún cuando es posible, cada usuario debe tener la

funcionalidad instalada a nivel cliente. Si uno solo no lo tiene, el sistema deja de funcionar de la manera esperada.

- *SourceSafe silenciosamente deja archivos muertos en el sistema:* Cuando se actualizan los archivos de trabajo locales al servidor, los archivos que fueron eliminados en el servidor deberían ser mostrados, o eliminados, ya que de cualquier modo los archivos se podrían restaurar del sistema de control de versiones. No hacer esto causa que archivos viejos o no requeridos se queden colgados del lado de un cliente, integrándose nuevamente al hacer *check-in*.
- *SourceSafe no funciona adecuadamente sobre redes lentas y por medio de Internet:* Se vuelve casi inusable sobre redes/conexiones lentas. Como SourceSafe funciona compartiendo sobre la red, si se publica un servidor de SourceSafe en la red se expone cualquier debilidad en la implementación del compartido de archivos que pueda tener el servidor.
- *Se vuelve extremadamente lento ver y regresar a versiones anteriores:* No es extraño llegar a necesitar una versión histórica del código fuente. Se puede necesitar una versión anterior para verificar un reporte de un *bug*, o regresar a una versión funcional. SourceSafe soporta esta función, sin embargo es bastante lento para proyectos grandes. Para obtener una versión histórica se requiere generar el historial para todo el proyecto, y luego especificar la versión seleccionando el último check-in que se necesita (por ejemplo, para un proyecto con cientos de archivos y un año de historia puede tomar más de 5 minutos).
- *Es difícil mantener copias múltiples de manera local de un mismo proyecto:* Al hacer cambios de manera masiva sobre un proyecto puede ser que se requiera hacer un cambio sobre la solución entera. La manera más eficiente de hacer esto es generar un duplicado para hacer el cambio. SourceSafe presenta un problema al hacer esto ya que solo reconoce una versión válida en el sistema. Se tendrá que hacer una copia del directorio físico, hacer el cambio, y darle la noción a SourceSafe de la ubicación de la copia.

Respecto a cuestiones de seguridad, también presenta determinadas desventajas:

- *SourceSafe se vuelve lento y problemático con proyectos grandes:* Microsoft recomienda que la base de datos de SourceSafe no exceda 5 GB en tamaño. Es común alcanzar esa capacidad en proyectos grandes, especialmente cuando se suben archivos binarios como documentos de Word, etc.
- *Integración de SourceSafe con Visual Studio puede causar inestabilidad:* SourceSafe se puede volver inestable cuando el sistema pierde conexión con la base de datos SourceSafe. Mientras que esto puede ser irritante al usar Visual SourceSafe de manera directa, se vuelve insoportable al usar Visual Studio con integración con SourceSafe ya que se puede generar perdidas.
- *SourceSafe depende de transferencia de archivos de manera peligrosa:* SourceSafe no corre realmente como un servidor si no que como un set de archivos compartidos a través de SMB. Como resultado se depende de que cada cliente no se comporte de manera errática. Una sola computadora que falle de hacer lo que debe puede corromper la base de datos. Un solo problema con el sistema para compartir archivos en una computadora puede inestabilizar la base de datos.
- *Se debe escanear a SourceSafe para detectar corrupción semanalmente:* Ante este riesgo grande de corrupción, Microsoft recomienda correr el programa de diagnostico "Analyze" de manera semanal. Mientras que Analyze está corriendo, todos los desarrolladores quedan encerrados fuera del sistema.
- *SourceSafe maneja zonas horarias múltiples de manera inadecuada:* Si se tienen equipos usando el mismo repositorio SourceSafe en diferentes zonas horarias es muy

probable que se tengan problemas. La única solución que propone Microsoft es ajustar la hora de la computadora entre zonas a una misma hora.

- *SourceSafe se corrompe:* El sistema de control de revisiones debe ser digno de confianza. Si los datos se corrompen el sistema es infuncional. El diseño fundamental de SourceSafe supone que los clientes funcionan adecuadamente todo el tiempo y nada se interpondrá entre la comunicación. Por lo tanto, SourceSafe es frágil y no es digno de confianza. Los datos se corrompen, se pierde el trabajo y se gasta tiempo en resolver el problema.

4.4. Darcs

Darcs es un sistema de control distribuido que fue diseñado para reemplazar los sistemas de control de versiones centralizados tradicionales como CVS y Subversion.

4.4.1. Características

Una característica que en su diseño lo distingue de CVS es que cada copia del árbol de código actúa como un repositorio completo, permitiendo que muchas versiones diferentes sean mantenidas en diferentes ubicaciones.

En el modo básico, los usuarios pueden obtener una copia de un repositorio, realizar cambios, registrar los cambios, deshacer cambios de otros repositorios y, finalmente, enviar cambios a otros repositorios. Para comenzar un nuevo repositorio, los usuarios pueden inicializar y crear cualquier directorio.

Otros repositorios pueden ser locales y accedidos por los protocolos SSH y HTTP. En estos casos se pueden enviar los cambios por mail, para los casos en que determinados usuarios no tengan permisos para acceso y/o modificación.

Darcs está escrito en el lenguaje de programación funcional Haskell.

4.4.2. Ventajas

- *La interfaz de Darcs es interactiva y bastante sencilla.* Muy pocos sistemas de control de versiones tienen una interfaz interactiva.
- *Simplicidad:* Se basa en un par de conceptos bastante sencillos y que son muy fáciles de entender.
- *No tiene servidor:* Para usar Darcs no hace falta ni instalar ni configurar ningún servidor. Con el cliente ya se puede trabajar:
- *Es muy fácil hacer proyectos derivados* (ramas personales). Esto es posible por su característica de ser distribuido.
- *Copias de trabajo y repositorios en Darcs*: El concepto básico sobre el que se asienta Darcs es que es lo mismo una "Copia de Trabajo" que un "Repositorio". Si una persona tienen un repositorio y otra persona quiere obtener una copia, esa persona automáticamente tendrá un repositorio y cuando guarde cambios en su copia de trabajo los cambios quedaran registrados en su repositorio y no en el repositorio original.

4.4.3. Desventajas

Darcs es criticado por motivos de *rendimiento.*

La peor de estas cuestiones fue la de la versión 1, donde el algoritmo de ordenación, en el peor de los casos, podría tener *orden exponencial*. En la versión 2 este defecto fue minimizado pese a que aun es posible que esto suceda.

4.5. Bazaar

El sistema de control de versiones Bazaar se convirtió en un proyecto GNU. Su meta es ser un sistema de control de versiones acorde a la colaboración y cooperación en el núcleo del software libre.

Bazaar es un sistema de control de versiones distribuido. Actualmente es uno de los más utilizados

4.5.1. Características

La operación básica para el control de versiones distribuido es el *branching* y *merging*, por lo que Bazaar está diseñado en gran parte para asegurarse que se pueda hacer branch y merge de forma eficiente entre individuos y equipos, usando ramas locales o remotas.

Para los desarrolladores, hay un API de extensión y una suite de plugins para permitirles extender Bazaar, y la biblioteca bzrlib facilita la tarea de incluir la funcionalidad de Bazaar en sus propias aplicaciones GPL.

Bazaar, busca facilitar el uso de control de versiones distribuido a través de todas las plataformas.

Esta escrito en Python.

4.5.2. Ventajas

- *Open Source:* Esta bajo la licencia GPL v2.
- *Amigable*: Focalizado en la usabilidad.
- *Inteligente:* Un buen soporte para renombrado de archivos y directorios.
- *Performance aceptable.*
- *Ligero:* No se necesita un servidor dedicado.
- *Extensible:* Se expone un API de Plugins.
- *Incrustable:* Soporta varios formatos de almacenamiento.
- *Tiene muchas unidades de prueba.*

***Bazaar es adaptable ya que se* ajusta a cualquier Workflow.**

- Solo
- Partner
- Centralizado
- Centralizado con *commits* locales
- Descentralizado con *mainline* compartido
- Descentralizado con revisión humana
- Descentralizado con revision automática

4.6. Plastic SCM

Plastic SCM es un Sistema de Configuraciones Software desarrollado por la empresa española Códice Software. Como objetivos fundamentales, Plastic trata de dar un mayor soporte al desarrollo paralelo, creación de ramas, fusión (merge) de ramas y seguridad.

4.6.1. Características y ventajas

- *Soporte de branching:* Para favorecer el desarrollo paralelo, Plastic se centra en dar soporte al branching, que consiste en dividir el desarrollo en distintas ramas, siguiendo una determinada política de uso, protección, desprotección, contenidos, etc. La principal diferencia entre el modelo de branching de Plastic y los implementados por sistemas tales como CVS, Perforce o Team Foundation Server estriba en que en lugar de realizar una copia de todo (o solamente de los metadatos) a cada nueva rama que se genera, las ramas son creadas como objetos vacíos. Solamente cuando un ítem es modificado, la nueva revisión creada es asignada a la rama. De este modo la rama

contiene solamente ficheros y directorios que se han modificado o creado con respecto a su rama padre. Este enfoque permite crear muchas ramas de forma sencilla. Debido al soporte que la infraestructura del modelo de branching de Plastic aporta, este sistema puede manejar miles de ramas en un solo repositorio sin pérdida notable en el rendimiento.

- *Directory versioning:* Tanto renombrado como cambios de ubicación de los directorios también están soportados.

Utilidades ventajosas y facilidades:
- *Ramas inteligentes:* Las ramas inteligentes consisten en ramas en las cuales el usuario puede definir una jerarquía de ramas, con soporte de los mecanismos de herencia. Una rama inteligente puede crearse a partir de una etiqueta específica, un conjunto de cambios (changeset) o una rama particular. La herencia de ramas puede ser configurada en la interfaz gráfica de usuario de Plastic. Pueden establecerse múltiples niveles de herencia a gusto del usuario.
- *Explorador de ramas:* El explorador de ramas es una interfaz de usuario que dibuja todas las ramas de un repositorio. Además representa las relaciones de fusión (merge) entre ramas y las relaciones de herencia. El explorador permite visualizar los cambios realizados en las ramas o en los conjuntos de cambios.
- *Árbol de revisiones en 3D:* El árbol de revisiones 3D muestra la evolución de un fichero o directorio dado, incluyendo enlaces de las operaciones de merge sufridas. Otro tipo de información incluida son las etiquetas y demás datos adicionales.

- *Seguimiento de merge:* Cada vez que se realiza una operación de merge, se crea un vínculo entre las revisiones origen y destino de la fusión.
- *Herramientas de merge y de diferencias:* Plastic incluye las siguientes herramientas: Diferencias en el código, Diferencias en imágenes, Merge de código, Merge binario, Diferencias y merge entre directorios.
- *Seguridad basada en Listas de Control de Acceso:* Cada objeto ubicado en un repositorio de Plastic tiene una Lista de Control de Acceso asociada. Se incluyen unos 25 tipos de permisos diferentes para permitir o denegar operaciones como por ejemplo protecciones, desprotecciones, fusiones, aplicación de etiquetas, creación de repositorios o espacios de trabajo, etc.
- *Base de datos de backend configurable:* Plastic soporta: MySQL, SQL Server, Firebird. Plastic utiliza tres tipos de bases de datos diferentes: Base de datos de repositorios (almacena información relativa a los repositorios alojados en un servidor determinado), Base de datos rep_xxx (una base de datos por repositorio), Base de datos de espacios de trabajo (almacena información relativa a los ficheros y directorios en los cuales trabaja el equipo de desarrollo).
- *Espacios de trabajo configurables:* Un espacio de trabajo es un directorio donde se mapean los contenidos del repositorio. Para seleccionar exactamente qué debe descargarse en el disco del usuario, cada espacio de trabajo tiene un selector asociado. Este selector proporciona múltiples posibilidades de personalización.

Otras ventajas:
- Plugins existentes para Eclipse, Visual Studio y JDeveloper.
- Permite la importación de todo el historial de proyectos que se encuentren bajo CVS, Subversion o SourceSafe

4.7. Mercurial

Mercurial es un sistema de control de versiones multiplataforma distribuido, para desarrolladores de software. Está implementado principalmente haciendo uso del lenguaje de programación Python, pero incluye una implementación binaria de diff escrita en C. Mercurial fue escrito originalmente para funcionar sobre Linux. Ha sido adaptado para Windows, Mac OS X y la mayoría de otros sistemas tipo Unix.

Mercurial es una herramienta de control de versiones de software que está teniendo un éxito importante entre proyectos muy conocidos como Mozilla, Xine y otros.

De todas maneras, diversos autores no lo recomiendan para proyectos grandes.

4.7.1. Características y ventajas

- Mercurial es, sobre todo, un *programa de línea de comandos*
- *Tiene buen rendimiento y buena escalabilidad*;
- *Permite el desarrollo distribuido*
- *Es distribuido y por lo tanto no necesita de un servidor;*
- *Posee una gestión robusta de archivos tanto de texto como binarios;*
- *Dispone de capacidades avanzadas de ramificación e integración.*

Para el acceso a repositorios mediante red, *Mercurial usa un protocolo eficiente, basado en HTTP,* que persigue reducir el tamaño de los datos a transferir, así como la proliferación de peticiones y conexiones nuevas. *Mercurial puede funcionar también sobre SSH.*

- *Incluye una interfaz Web integrada.*
- *Es fácil de acondicionar.*
- *Permite migraciones desde otro sistema de control de versiones*

4.7.2. Desventajas

- El *aprendizaje* puede demandar mucho tiempo: Varios usuarios notaron esta deficiencia.
- *Interfaz deficiente*
- *Dificultad para la eliminación de ramas.*

4.8. Git

Git es un software de sistema de control de versiones, pensado para la eficiencia y confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número archivos de código fuente. Git se ha convertido en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo del sistema operativo GNU/Linux.

4.8.1. Características y ventajas

- El diseño de Git se basó en BitKeeper y en Monotone.
- *Fuerte apoyo al desarrollo no-lineal, por ende rapidez en la gestión de ramificaciones y mezclado de diferentes versiones:* Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no-lineal. Una presunción en Git es que un cambio será fusionado o empalmado mucho mas frecuentemente de lo que se escribe originalmente, conforme se pasa entre varios programadores que lo revisan.
- *Gestión distribuida:* Al igual que Darcs, BitKeeper, Mercurial, SVK, Bazaar y Monotone, Git le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales. Los cambios se importan como ramificaciones adicionales y pueden ser fusionados en la misma manera que se hace con la ramificación local.
- *Los almacenes de información pueden publicarse por HTTP, FTP, rsync* o mediante un protocolo nativo, ya sea a través de una conexión *TCP/IP simple o a través de cifrado SSH.* Git también puede emular servidores CVS, lo que habilita el uso de clientes CVS pre-existentes y módulos IDE para CVS pre-existentes en el acceso de repositorios Git.
- Los repositorios *Subversion y svk* se pueden usar directamente con git-svn.
- *Gestión eficiente de proyectos grandes,* dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución.

- Todas las versiones previas a un cambio determinado, implican la *notificación de un cambio posterior* en cualquiera de ellas a ese cambio (denominado autenticación criptográfica de historial). Esto existía en Monotone.

Resulta algo más caro trabajar con ficheros concretos frente a proyectos, eso diferencia el trabajo frente a CVS, que trabaja en base a cambios de fichero, pero mejora el trabajo con afectaciones de código que concurren en operaciones similares en varios archivos.

- *Los renombrados se trabajan en base a similitudes entre ficheros, aparte de nombres de ficheros*, pero no se hacen marcas explícitas de cambios de nombre en base a supuestos nombres únicos de nodos de sistema de ficheros, lo que evita posibles conflictos por coincidencias de ficheros diferentes en un único nombre.

Git realiza realmacenamiento periódico en paquetes (ficheros): Esto es relativamente eficiente para escritura de cambios y relativamente ineficiente para lectura si el reempaquetado (en base a diferencias) no ocurre cada cierto tiempo.

- *Es portable para varios SO.*

4.9. ClearCase

Rational ClearCase es una herramienta de software para control de versiones (gestión de configuración, SCM) de código fuente y otros recursos de desarrollo de software. Es desarrollado por la división de Rational Software de IBM. Según IBM, ClearCase puede ser utilizado por grandes y medianas empresas y puede manejar proyectos con muchos desarrolladores.

4.9.1. Características y ventajas

- *Construir la auditoría:* El sistema de archivos de red proporcionados por MVFS permite la construcción de la auditoría. Permite monitorear y registrar las operaciones de E/S realizadas durante el proceso de construcción. La construcción de auditorías se realiza con las herramientas de línea de comandos.
- Tiene la capacidad de brindar el *control de versiones* de todos los assets de desarrollo. Versiona cada asset del ciclo de vida del desarrollo de software. Proporciona seguimiento a los cambios a cada archivo y directorio, manteniendo una historia completa de versiones de assets, incluyendo código fuente, binarios, ejecutables, documentación, test scripts y librerías. Los desarrolladores pueden volver a cualquier configuración anterior de build o baseline para identificar que versiones de que archivos dispararon un cambio determinado.

El versionado de Rational ClearCase incluye:
- *Modelo de desarrollo Checkin/checkout*
- *Versionado de directorios, subdirectorios* y todos los objetos del file system
- *Base de datos de objetos versionados segura*, que garantiza que solo usuarios acreditados puedan acceder a los archivos y solo a aquellos a los que los habilita su perfil
- Conversión de archivos de TeamConnection y CMVC, Merant PVCS, Microsoft Visual SourceSafe, RCS, CVS y SCCS

- *Permite el desarrollo en paralelo:* Acelera el desarrollo en equipo a través de un soporte para el desarrollo en paralelo. El branching es automático y permite a los desarrolladores diseñar, codificar, testear y optimizar el software eficientemente a partir de un código base en común. Mediante tecnología de diff/merge, ClearCase acepta automáticamente los cambios que no ocasionan problemas, y resalta los conflictivos para una resolución inmediata por parte del desarrollador/integrador.
- *Provee la administración de workspace incluyendo el soporte de vistas dinámicas:* ClearCase provee automáticamente workspaces de desarrollo. Los desarrolladores pueden comenzar a trabajar rápidamente simplemente uniéndose al proyecto y seleccionando una actividad. Se mantienen automáticamente las áreas de trabajo de los desarrolladores con el conjunto adecuado de versiones de los archivos. Proporciona vistas a los desarrolladores de las versiones exactas de los archivos

requeridos para completar una tarea específica, aislándolos de los cambios potencialmente desestabilizantes que pudieran efectuar otros miembros del equipo. Los desarrolladores pueden seleccionar uno de los dos tipos de vistas optimizadas para modelos de uso local o en red.

- *Utiliza el Unified Change Management* (UCM, el proceso configurable, basado en actividades para la administración del cambio). UCM, según IBM, es "el resultado de las experiencias en proyectos de desarrollo de software de cientos de usuarios Rational en miles de proyectos". Proporciona un modelo de proceso de SCM que permite que los equipos, pequeños y grandes, puedan iniciar rápidamente nuevos proyectos, definir procesos de cambio, y automatizar consistentemente actividades claves del SCM. Con SCM y ClearCase los equipos automatizan y aceleran los pasos requeridos para crear y mantener las áreas de trabajo de los desarrolladores y para crear y administrar baselines de componentes
- *Puede escalar de equipos de mediano a largo porte* ofreciendo soporte de plataformas incluyendo Windows, Linux y UNIX.
- *Proporciona administración avanzada de builds y auditoria*. Detecta automáticamente las dependencias y produce un inventario detallado que lista las versiones de archivo exactos que componen un build, Provee reportes claros sobre los archivos versionados que componen cada build remoto (Esta capacidad permite recrear builds a pedido), entre otras características.
- Por medio del Rational ClearCase MultiSite, *brinda soporte a equipos geográficamente distribuidos.*

Se integra con:
- IDEs líderes (WebSphere Studio, Microsoft .NET).
- Rational Suite y Rational Team Unifying Platform.
- Herramientas para desarrollo Web y authoring.

4.9.2. Desventajas

- *Las transacciones no son atómicas:* los cambios en archivos o directorios son independientes de los demás, a diferencia de algunos otros sistemas en donde múltiples cambios se pueden cometer al mismo tiempo en forma atómica
- *Velocidad:* las vistas dinámicas son más lentas que la de los sistemas de archivos locales, incluso con una buena infraestructura de red. Debido a que MVFS requiere acceso al servidor cada vez que se accede a un archivo, el rendimiento del sistema de archivos depende de la capacidad del servidor.
- *Velocidad en los clientes de Windows:* el rendimiento de disco para los clientes de Windows es extremadamente bajo.
- *La sensibilidad a los problemas de red:* No es posible trabajar con vistas dinámicas sin conexión.

4.10. Team Foundation Server

Team Foundation Server (TFS comúnmente abreviado) es un producto de Microsoft, que integra un sistema de control de versiones, control de código fuente, colecciones de datos, reportes y seguimiento de proyectos, y está destinado a proyectos de desarrollo colaborativo de software. Está disponible como software independiente o del lado del servidor utilizando la plataforma para Visual Studio Team System (VSTS).

4.10.1. Arquitectura y Características

Team Foundation Server funciona en una arquitectura de tres capas: la capa del cliente, la capa de aplicación y la capa de datos.

La *capa del cliente* se utiliza para crear y gestionar proyectos y para acceder a los elementos que son almacenados y manejados por el proyecto. TFS no incluye ninguna interfaz de

usuario para esta capa, sino que provee servicios Web que las aplicaciones cliente pueden utilizar para integrar la funcionalidad de TFS para estas aplicaciones. Estos servicios Web son utilizados por aplicaciones como Visual Studio Team System para utilizar TFS como almacenamiento de datos back-end o como aplicación de gestión dedicada TFS, como el incluido Team Foundation Client.

Los servicios Web son de la *capa de aplicación*. La capa de aplicación también incluye un portal Web y un repositorio de documentos facilitados por Windows SharePoint Services. El portal Web, (Team Project Portal), actúa como el punto central de comunicación para proyectos gestionados por TFS. El repositorio de documentos se utiliza tanto para los elementos del proyecto, como para el monitoreo, así como para los datos agregados y los informes generados.

La *capa de datos* (se recomienda SQL Server 2005 Standard Edition) ofrece almacenamiento persistente de datos para el repositorio de documentos. La capa de datos y de aplicación puede existir en diferentes servidores físicos o virtuales. Así, la capa de datos no está expuesta a la capa de cliente.

La mayoría de las actividades de Team Foundation Server giran en torno a los "elementos de trabajo". Los elementos de trabajo son una única unidad de trabajo que necesita ser completada. Un elemento de trabajo tiene campos para definir el área, la iteración, el apoderado, reportes, historiales, archivos adjuntos, y varios otros atributos. Los elementos de trabajo en sí pueden ser de varios tipos, tales como un error, una tarea, una evaluación del servicio de calidad, un escenario, y así sucesivamente. Se establece qué tipos de elementos de trabajo están disponibles y qué atributos contiene cada tipo de elemento de trabajo. Estos elementos se almacenan internamente en formato XML, y su esquema se puede personalizar para agregar otros atributos a diferentes elementos, o crear nuevos elementos en función de cada proyecto. Cada elemento de trabajo esta asociado a políticas de control que especifican quienes puede acceder o modificar los elementos. También incluye registro de creación, acceso, modificación, etc. y, opcionalmente, notificación a usuarios determinados cuando se producen determinados eventos.

Se puede contener uno o más proyectos de equipos, que se componen de soluciones de Visual Studio, archivos de configuración de Team Build y Team Load Test Agents, y un único repositorio que contiene los documentos pertinentes para el proyecto.

Un proyecto de equipo contiene los elementos de trabajo definidos por un usuario, los branch origen, y los informes que serán gestionados por TFS.

TFS ofrece capacidades para la gestión de estos proyectos. Al crear un proyecto, debe elegirse un entorno para el desarrollo del software, que *no podrá cambiarse después*. TFS incluye varios templates para los más comunes, *incluidas las metodologías ágiles y formales.*

> El estado de algunos elementos del proyecto se puede configurar para que se actualicen automáticamente, cuando se actualizan los elementos de trabajo. TFS se puede integrar con Microsoft Excel para la creación y seguimiento de los elementos de proyecto.
>
> El estado de los elementos pueden crearse y editarse en un documento de Excel y la hoja de cálculo resultante puede ser importada a TFS. TFS también puede integrarse con Microsoft Project como el proyecto de gestión.
>
> Los elementos del proyecto también se pueden exportar como documentos de Excel para el posterior análisis de los datos.

TFS no incluye una interfaz de usuario para realizar todas estas tareas. Las capacidades están expuestas a través de servicios Web, que luego son utilizados por aplicaciones cliente como Visual Studio Team System IDE. Sin embargo, TFS incluye un cliente de Team Foundation (TFC), aplicación que puede ser utilizada para realizar estas tareas fuera del IDE de VSTS. TFC también opera mediante la invocación de servicios Web de la misma. TFS expone una API de cliente que pueden ser utilizados por aplicaciones cliente para acceder a la funcionalidad.

TFS puede integrarse con más aplicaciones.

En la figura 4.2. puede verse gráficamente la arquitectura de TFS.

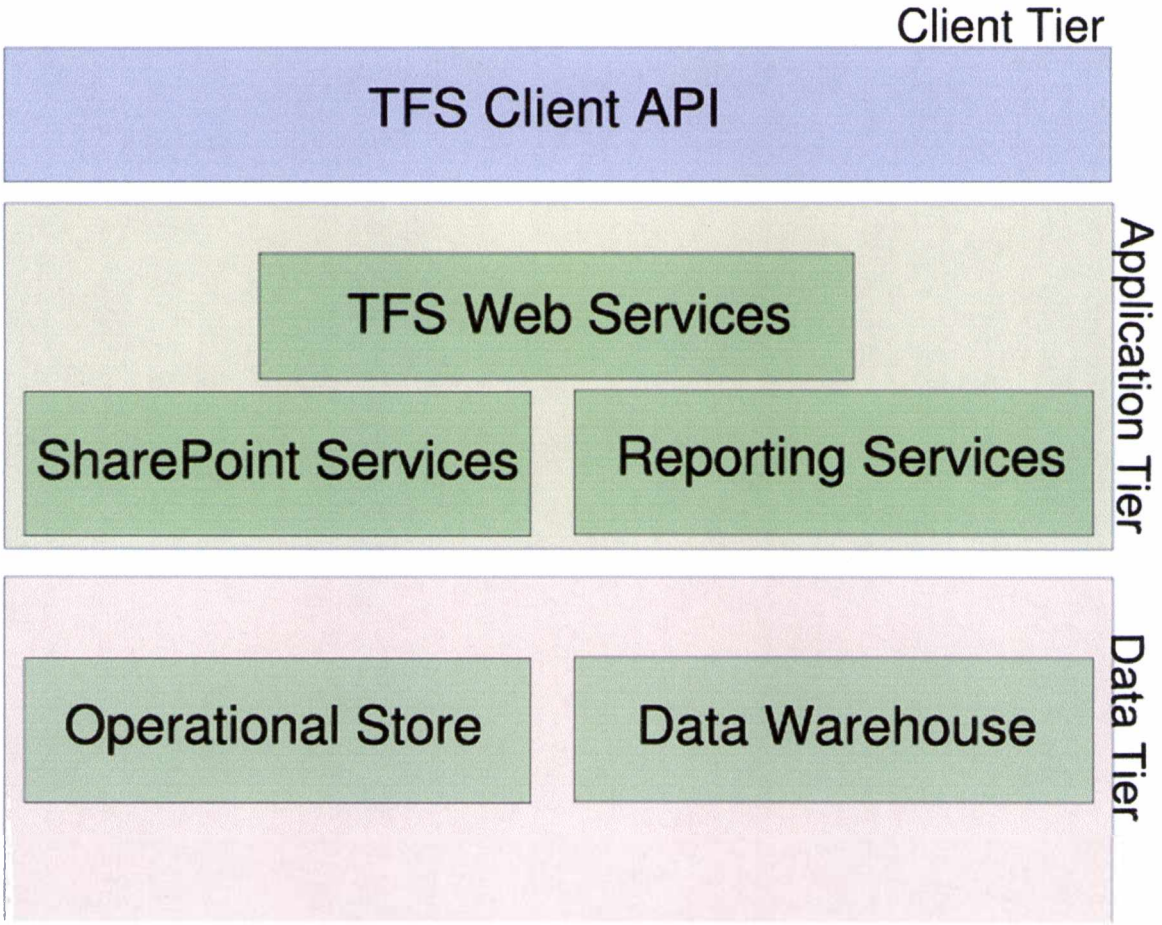

Figura 4.2. Arquitectura de TFS

4.10.1.1. Control de código fuente

Team Foundation Server proporciona un repositorio de control de código fuente, llamada Team Foundation Version Control (TFVC). TFVC almacena todo el código fuente, el registro de todos los cambios y check-outs actuales en una base de datos de SQL Server. Es compatible con funciones como la verificación simultánea de varios check-outs, la resolución de conflictos, retomar cambios aplazados, branching y merging, y la posibilidad de establecer niveles de seguridad en cualquier nivel del árbol del código fuente. El repositorio de control de código fuente no permite diversas referencias a un mismo archivo desde distintos directorios para que apunte a diferentes versiones.

TFVC apoya el branching en todo el nivel de código fuente, así como el de archivos y directorios individuales, para que cada rama se pueda mantener individualmente. Múltiples ramas se puede combinar, conciliando automáticamente las diferencias o marcándolas para la inspección manual si no se puede hacer.

4.10.1.2. Presentación de informes

La presentación de informes es otro componente importante de Team Foundation Server. Utilizando los datos combinados de los elementos de trabajo, el conjunto de cambios, la información proporcionada por el Team Build y los resultados de los agentes de pruebas, puede crearse una variedad de informes (por ejemplo, la tasa de cambio de código a través del tiempo, las listas de errores que no tienen casos de prueba, etc.). Los informes se construyen utilizando SQL Server Reporting Services, y pueden ser exportados en diferentes formatos, incluyendo Excel, XML, PDF y TIFF. Los informes se pueden acceder tanto a través de Visual Studio, como a través del portal Web.

4.10.1.3. Portal de proyecto

En función de cada proyecto, TFS también crea un sitio, que puede ser utilizado para seguir el progreso del proyecto, así como para explorar los elementos de trabajo y los documentos de control de código fuente del proyecto. También puede ser usado para ver los informes generados. Los usuarios asociados pueden utilizarlo para comunicarse entre sí. Los comentarios pueden estar relacionados con diversos temas. Para cada proyecto, en función de las propiedades del proyecto, TFS utiliza una plantilla predefinida que define la apariencia del sitio. Estas plantillas pueden ser personalizadas por el administrador de TFS.

4.10.1.4. Servicios compartidos

TFS ofrece algunos servicios que pueden ser utilizados para la integración con otras aplicaciones como IDEs y sistemas de gestión de proyectos. El *Linking service* permite crear una relación entre elementos. Los servicios de seguridad permite la creación de grupos de seguridad entre usuarios, a los que se asignan diversos derechos de acceso. El servicio de clasificación permite la definición de políticas para clasificar los elementos de forma automática en base a varios criterios. Y el servicio de eventos permite que determinados componente disparen eventos y genera una notificación asignada al evento.

4.10.1.5. Team Build

Team Build es un servidor de compilación que esta incluido en TFS que puede instalarse en casi cualquier máquina que soporte Visual Studio. Las máquinas configuradas con Team Build pueden ser utilizadas por desarrolladores para hacer una construcción completa de las versiones más recientes del software incluidas en el control de código fuente. Los registros de cada Build (tenga éxito o fracase), se mantienen de manera que los desarrolladores y los administradores pueden crear un seguimiento de los avances del proyecto. Si la compilación es correcta, se analizan los cambios en el control de código fuente desde la última compilación correcta, y actualiza a los elementos de trabajo para indicar que se ha avanzado. Un probador puede confirmar o negar que el error se haya resuelto. Team Build es altamente personalizable.

Capítulo 5

Plataforma Jazz. Rational Team Concert

Este capitulo comienza a abordar la temática principal de la tesina. En este apartado se describen las características, arquitectura y especificaciones técnicas de la plataforma Jazz, así también como los objetivos que IBM busca alcanzar con ésta, software y plugins soportados, entre otros.

Finalmente se aborda en particular el software RTC, se especifica detalladamente aspectos técnicos, características, objetivos, ventajas, entre otros, como fundamentos por los cuales IBM recomienda el uso y promoción de su software.

Cabe aclarar que este capitulo busca principalmente exponer la información proporcionada por IBM acerca de RTC. A partir de la experiencia realizada, y según las conclusiones arribadas, a modo crítico se podría apoyar o refutar dicha información (total o parcialmente), explayar características, plantear interrogantes, discutir información, etc.

5.1. Plataforma Jazz

La iniciativa Jazz dentro de IBM Rational es un programa de modernización de las herramientas del ciclo de vida de desarrollo.

El objetivo de Jazz es proporcionar soporte a equipos de desarrollo cada vez más dinámicos y distribuidos en un ámbito global mediante un entorno colaborativo y abierto. Tradicionalmente tanto IBM Rational como sus clientes invierten un gran esfuerzo en integrar las distintas herramientas de soporte al ciclo de vida de desarrollo: a medida que el número de herramientas aumentan también el coste de esta integración (Figura 5.1).

Jazz es una nueva plataforma de tecnología de IBM Rational para la entrega de software basada en la colaboración. La plataforma Jazz está adaptada de forma exclusiva a los equipos globales y distribuidos, y diseñada para fomentar el trabajo conjunto de creación de

software de manera más productiva, transparente y basada en la colaboración. Puede imaginarse la tecnología Jazz como una infraestructura ampliable que se integra de forma dinámica y sincroniza personas, procesos y activos asociados a los proyectos de desarrollo de software.

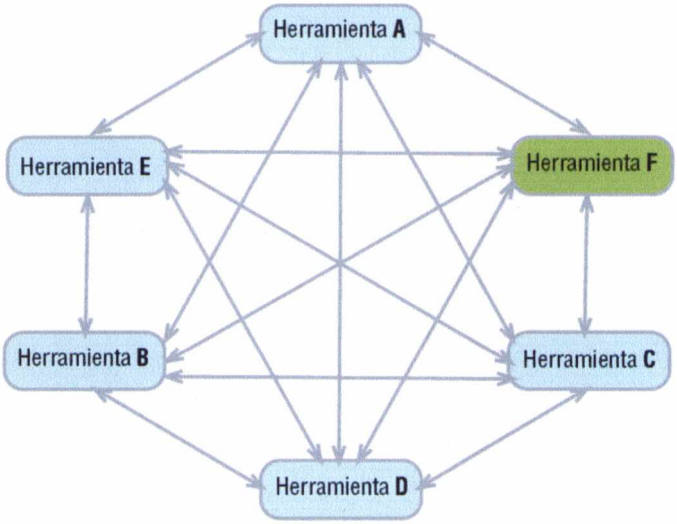

Figura 5.1. Integración en Jazz

Jazz proporciona un middleware de integración (Jazz Server) de forma que las distintas herramientas se comunicarán enviando o recibiendo mensajes a dicho middleware. A su vez el Jazz Server proporcionará servicios de colaboración comunes que asegurará el control efectivo del ciclo de vida de desarrollo (Figura 5.2).

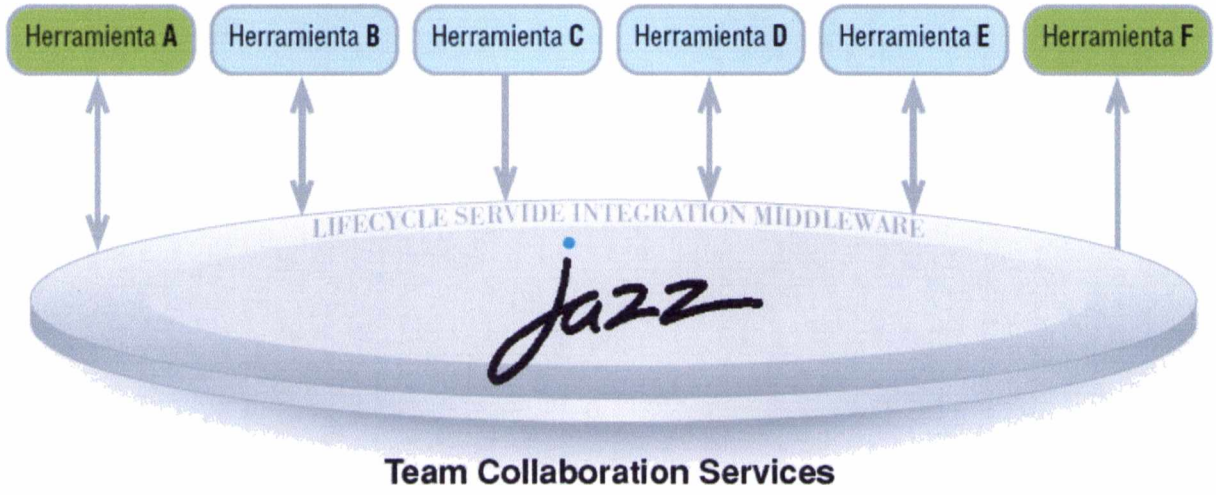

Figura 5.2. Middleware de integración en Jazz

La tecnología Jazz permite la colaboración, no sólo entre profesionales de software, sino también entre accionistas empresariales, expertos en la materia y todas las personas relevantes que participan en la entrega satisfactoria de software. Esto aumenta de forma revolucionaria el valor de los equipos de entrega de software, ya que les permite convertir el software en la base para impulsar la innovación en toda la empresa.

Jazz es una plataforma de tecnología, no un producto. Las ofertas de producto que se basen en la plataforma Jazz disfrutarán de una amplia gama de funciones para el desarrollo y la distribución de software en equipo. La familia *Rational Team Concert*, *Rational Quality Manager* y *Rational Requirements Composer* son las primeras ofertas basadas en tecnología Jazz.

5.1.1. Principios

Los principios fundamentales sobre los que se ha construido la plataforma Jazz:

- Proporcionar integración de tareas a lo largo del ciclo de vida.
- Plataforma existente, abierta y basada en tecnologías actuales.
- Soporte a pequeños y grandes equipos distribuidos geográficamente.
- Foco en el equipo y no en fases del ciclo.

5.1.2. Visión general

Con los años, el desarrollo de software ha sido comparado con otras disciplinas (con el arte, la ciencia, e incluso con procesos de fabricación).

El software es mejor desarrollarlo por un equipo de personas trabajando juntas, que reaccionen y respondan conjuntamente a fin de lograr el mejor resultado.

JAZZ es una iniciativa de IBM para ayudar a los equipos a desarrollar y entregar software eficazmente.

Inspirado por artistas que transformaron la expresión musical, JAZZ es una iniciativa que transforma el desarrollo y la entrega de software haciéndola *colaborativa, productiva y transparente*.

La iniciativa JAZZ está compuesta de tres elementos fundamentales:

- *Una arquitectura para la integración en el ciclo de vida*
- *Un conjunto de productos diseñados para priorizar al equipo (Team)*
- *Una comunidad de Stakeholders (partes interesadas)*

5.1.2.1. Una arquitectura para la integración en el ciclo de vida

Los productos basados en JAZZ incorporan un enfoque innovador de integración basado en ser una arquitectura abierta, servicios flexibles e Internet. A diferencia de los productos cerrados, JAZZ es una plataforma abierta diseñada para soportar a cualquier tipo de participante de una organización que quieren mejorar el ciclo de vida de un software e integrar diversas herramientas.

La arquitectura integrada de JAZZ está diseñada para dar a las organizaciones la flexibilidad para lograr un ambiente ideal para la entrega de un software, utilizando las herramientas y proveedores preferidos. Esta flexibilidad permite además, adaptar y evolucionar tanto al entorno como las necesidades lo requieran, y avanzar a su propio ritmo. La arquitectura integrada de JAZZ define un conjunto común de servicios de que pueden ser aprovechadas por cualquier herramienta de JAZZ, y explica la forma de acceder y utilizar dichos servicios.

5.1.2.2. Un conjunto de productos diseñados para priorizar al equipo (Team)

El conjunto de productos de Jazz consiste en una plataforma común y un conjunto de herramientas que permiten a todos los miembros del equipo de desarrollo colaborar más fácilmente. Esto refleja que la visión central del desarrollo de software no es ni el individuo ni el proceso... es la colaboración del equipo. Los productos más importantes son (Figura 5.3.):

- Rational Team Concert
- Rational Quality Manager
- Rational Requirements Composer

Productos y soporte

IBM ha decidido que Jazz sea una iniciativa abierta a toda la comunidad de desarrollo y para ello desde fases tempranas del programa ha aportado e intercambiado información con la comunidad a través de www.jazz.net.

Clientes y proveedores se beneficiarán de esta transparencia tal como sucede con Eclipse. IBM Rational introducirá de forma paulatina el soporte de Jazz en su actual oferta a la vez que se lanzarán nuevas ofertas basadas en esta tecnología.

Los nuevos productos que IBM Rational ha lanzado al mercado beneficiándose de los servicios proporcionados por esta nueva plataforma tecnológica son:

- ***Rational Team Concert:*** Solución para la gestión unificada del cambio, control de versiones y compilación que permite que diferentes equipos colaboren en tiempo real en el desarrollo de software. Incluye soporte para la gestión de compilaciones, control de orígenes y elementos de trabajo, además de funciones de unificación de los equipos del entorno Jazz. Proporciona un entorno de desarrollo de software flexible y de colaboración para desarrolladores, arquitectos y gestores de proyectos. *Rational Team Concert* simplifica el intercambio de información directamente en el contexto del trabajo que se esté desempeñando (se informan automáticamente cambios, solicitudes, y/o cualquier tipo de modificación). *Rational Team Concert* está disponible con integraciones listas para utilizar que facilitarán la interoperatividad con productos de gestión de cambios de Subversion, Build Forge, Rational ClearCase y ClearQuest. Dichas integraciones están pensadas para ampliar la flexibilidad y proporcionar pruebas eficaces en equipos reducidos dentro de un mayor entorno empresarial.
- ***Rational Quality Manager:*** Gestión unificada de todas las pruebas a través de una consola Web centralizada. Proporciona una solución personalizable y de colaboración para planificar pruebas y casos de pruebas, implementar pruebas manuales, ejecutar pruebas, analizar e informar resultados, controlar el flujo de trabajo, y realizar seguimientos e informes de métricas, capaz de cuantificar el impacto de las decisiones de los proyectos y de las entregas y adaptarse a los objetivos empresariales.. Integración con herramientas de automatización de pruebas funcionales (Rational Functional Tester), de carga (Rational Performance Tester) y de seguridad de aplicaciones Web (Rational AppScan).
- ***Rational Requirements Composer***: Solución para ayudar a los usuarios y analistas en la definición de los requisitos y mejorar la comunicación entre usuarios de negocio y el equipo de desarrollo: edición y revisión centralizada de documentación relacionada con los requisitos (faxes, imágenes, documentos Word, presentaciones...), prototipado de interfaces de usuario, definición de los procesos de negocio, glosario centralizado de términos. Así mismo, Rational Requirements Composer se integra con la herramienta de gestión de requisitos, IBM Rational RequisitePro, de forma que los requisitos identificados por los usuarios y analistas puedan ser automáticamente visibles y gestionados por el equipo de desarrollo, permitiendo así la trazabilidad con otras fases del ciclo de vida de desarrollo. *Rational Requirements Composer* es una solución de definición de requisitos que permite a las organizaciones mejorar los procesos de requisitos con funciones de descubrimiento y definición fáciles de utilizar. Al proporcionar varias herramientas visuales y de colaboración, Requirements Composer permite capturar y refinar las necesidades empresariales en requisitos claros, y mejora la calidad, la velocidad y la alineación en todo el ciclo de vida de IT. Visualizar los resultados antes de utilizar los recursos evita los costosos errores de requisitos y las correcciones en los proyectos.

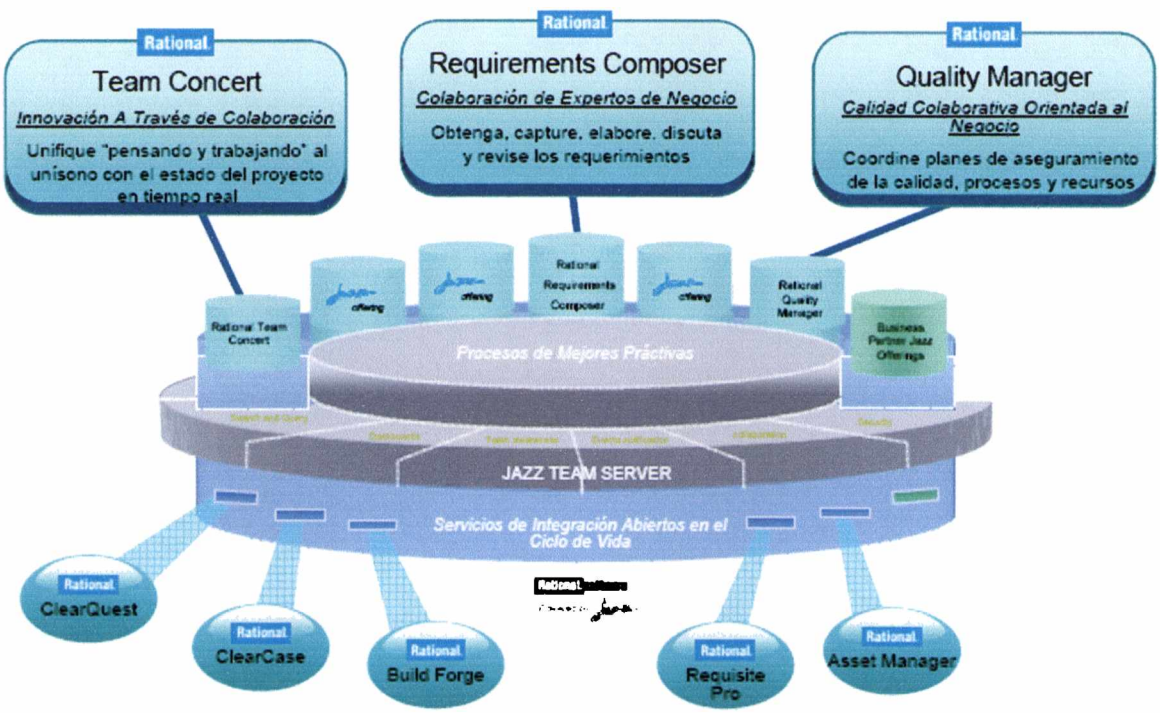

Figura 5.3. Productos integrados en Jazz

5.1.2.3. Una comunidad de Stakeholders (partes interesadas)

Jazz no es sólo un software tradicional donde un desarrollador ayuda a otro desarrollador. Involucra a todos los participantes, como los clientes y otras partes interesadas, mediante una conversación directa, temprana y continua.

Una vez que se une un participante, puede comunicarse con los equipos de desarrollo, ver el historial del proyecto, recibir feedback sobre lo que funciona y lo que no, y seguir los defectos y solicitudes de mejora. También se tiene una visibilidad completa de planes detallados, estados y el progreso. Se pueden utilizar también los productos mediante interfaces Web. El beneficio de esta transparencia es que se permite formar parte de un feedback continuo sobre las decisiones que se toman durante el desarrollo. Al ofrecer una respuesta temprana y frecuente, se puede entender e influir sobre el desarrollo y las prioridades antes de que estas decisiones sean plasmadas.

5.1.2.4. Objetivo

El objetivo es proporcionar un entorno de trabajo de fricción que ayuda a los equipos a colaborar, innovar y crear un gran software. Para tal fin, Jazz impulsa mejoras fundamentales en la colaboración en equipo, la automatización, y la presentación de informes a través del ciclo de vida del software.

5.1.2.5. Colaboración

Las herramientas de Jazz reflejan la idea de que el centro de desarrollo de software no es, ni el individuo, ni el proceso... es la colaboración dentro de un equipo. También se reconoce que el equipo se extiende más allá de los profesionales en el desarrollo, para incluir dentro del proyecto a todas las partes interesadas. Uno de los objetivos de la iniciativa Jazz es permitir la transparencia de los equipos y proyectos para una colaboración continua y sensible al contexto que puede:

- Promover la innovación
- Construir cohesión del equipo
- Aprovechar el talento dentro y fuera de la empresa

5.1.2.6. Automatización

Uno de los objetivos de la iniciativa Jazz es automatizar procesos, flujos de trabajo y tareas para que las organizaciones puedan adoptar los principios de desarrollo más propicios para estas.

La iniciativa JAZZ se esfuerza en:

- Mejorar el soporte y la ejecución de cualquier proceso, incluidos los procesos ágiles
- Reducir lo tedioso y el tiempo consumido de las tareas manuales
- Capturar información sobre avances, eventos, decisiones y autorizaciones sin la entrada de datos adicionales

5.1.2.7. Reporting (Presentación de informes)

Conseguir un acceso rápido a información estadística y basada en los hechos es esencial para cualquier proyecto. Pero con demasiada frecuencia, plasmar esta información resulta tedioso porque generalmente se genera de forma manual y se entrega fuera del tiempo en la que es necesaria. La iniciativa de Jazz se centra en ofrecer una visión en tiempo real en los programas, proyectos y utilización de los recursos para ayudar a los equipos:

- Identificar y resolver problemas en etapas tempranas del ciclo de vida del software
- Obtener métricas basadas en hechos (no estimaciones) para mejorar la toma de decisiones
- Aprovechar las métricas para una continua mejora de la capacidad individual y grupal.

5.1.3. Arquitectura de Jazz

Jazz se basa en los principios arquitectónicos que representan un cambio fundamental de los enfoques adoptados en el pasado.

En conjunto, estos enfoques permiten a los equipos "navegar por una Web colaborativa" para acceder a equipos, procesos y artefactos.

Algunas ideas claves:

- Jazz separa la implementación de las herramientas de la definición del acceso a los datos. La semántica de datos no se basan en los "conocimientos secretos" embebidos en el código del producto.
- Los datos en JAZZ se asocian a través de bases de datos independientes, mediante protocolos de Internet. Jazz no asume que todos los datos están en una base de datos única.
- Jazz puede acceder e integrar datos donde estén; Jazz no necesita importar y exportar datos entre las herramientas o repositorios
- Jazz asume un modelo de datos abierto, flexible y distribuido. Jazz no asume que hay un único modelo de datos que es gestionado de forma centralizada, ni que cada herramienta tiene que entender el modelo de datos completo con el fin de participar.
- Jazz permite a las herramientas que sean implementadas en cualquier lenguaje de programación o plataforma que contemple las redes (Lan, Internet).
- Jazz soporta múltiples tecnologías cliente. Mientras que las interfaces de usuario Web son la mejor opción para muchas herramientas de Jazz, Jazz también soporta clientes basados en Eclipse y Microsoft Visual Studio.

En la figura 5.4. se muestra gráficamente la arquitectura de Jazz

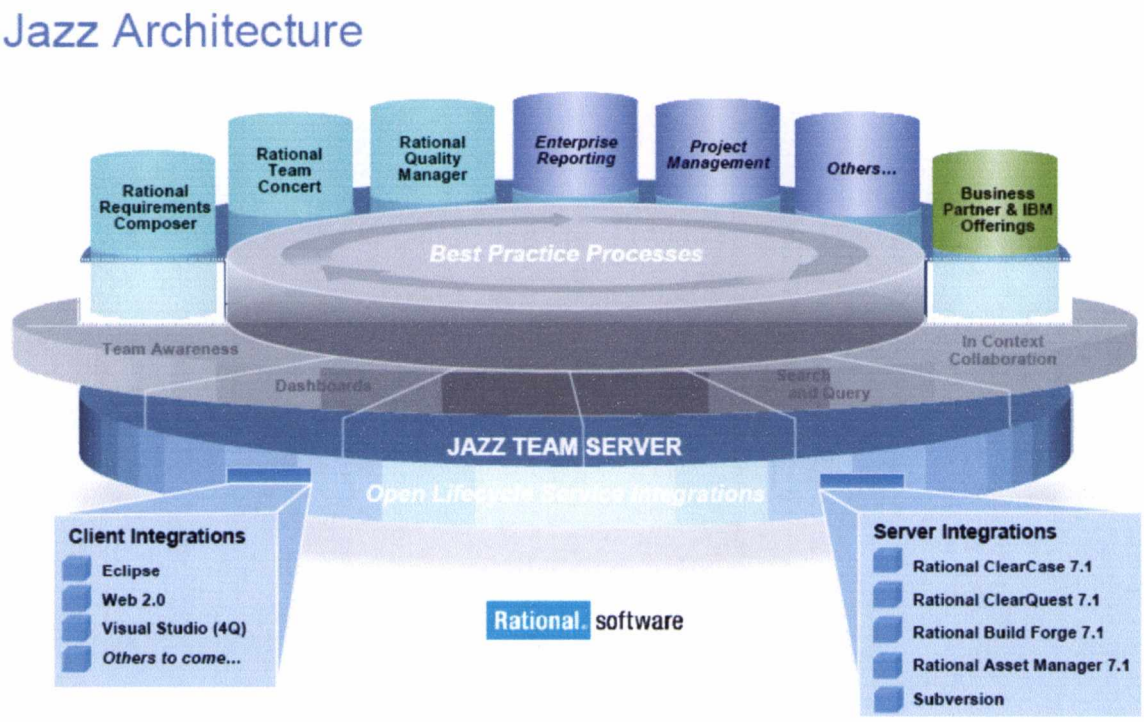

Figura 5.4. Arquitectura Jazz

5.1.3.1. Open Services for Lifecycle Collaboration (OSLC)

Open Services for Lifecycle Collaboration (OSLC) es una iniciativa para permitir la interoperabilidad de las herramientas y recursos a través de los diferentes proveedores. Hoy en día, los clientes utilizan herramientas de múltiples proveedores para capturar y gestionar datos de todo el ciclo de vida del software. Estos datos se suelen acceder mediante el uso de la herramienta que los crean o mediante el acceso a una base de datos

utilizada por la herramienta. La falta de protocolos comunes entre recursos hace difícil la gestión de estos datos.

5.1.3.2. Jazz Integration Architecture (JIA)

Sobre la base de OSLC, la arquitectura integrada de Jazz (Jazz Integration Architecture - JIA) permite un mayor nivel de integración que se puede lograr sólo con OSLC. La arquitectura tiene dos facetas principales:

1. La JIA define un conjunto de servicios (llamados Jazz Foundation Services) que pueden utilizar las distintas herramientas, incluidos la administración de usuarios y proyectos, seguridad, colaboración, consulta, y otras capacidades comunes. Estos servicios permiten a las herramientas implementar sus propios servicios de modo que se puedan combinar con otras herramientas. En la práctica, una herramienta existente puede optar por adoptar estos servicios de forma selectiva.
2. Las herramientas definen sus servicios específicos con un lenguaje neutro, para que los clientes puedan acceder a ellos. El OSLC proporciona un lugar para este tipo de especificaciones de servicios para que sean conjuntamente desarrollados, compartidos y evolucionen en un ambiente neutro de un proveedor.

5.1.3.3. Implementación de Jazz

En el núcleo de la JIA (Jazz Integration Architecture) está el Jazz Team Server (JTS). El JTS implementa los servicios (Jazz Foundation Services) descriptos por la JIA, que permiten a los grupos de herramientas a trabajar juntos. Estos servicios incluyen la administración de usuarios y proyectos, seguridad, colaboración, consulta y otros servicios genéricos. Cuando se instalan, las herramientas son asociadas y trabajan en conjunto con un JTS particular. Jazz Team Server puede constar de uno o más servidores físicos que actúan conjuntamente como un único servidor lógico.

Para facilitar la implementación de productos que adhieran a la JIA, se proporcionan toolkits de Jazz. Se incluyen bibliotecas de lenguajes específicos para acceder a varios servicios de Jazz Foundation Services y a servicios en particular; Se ofrecen clientes Eclipse, Web y por línea de comandos. Los Framework de Jazz se consideran que son más productivos que esenciales; la integración de JIA es alcanzable por la mayoría de los lenguajes de programación capaces de procesar HTTP y XML. Esto significa que los Framework de Jazz podrían ser desarrollados por cualquier persona en cualquier lugar.

La Fundación Jazz (Jazz Foundation) refiere a las implementaciones de los Jazz Team Server incluidos los servicios de Jazz Foundation Services, desarrollados y mantenidos en Jazz.net. También se incluyen un conjunto de Framework específicos de Jazz que ayudan en la construcción de productos de Jazz con diversos grados de integración en JIA.

5.1.4. C/LAM

¿Qué es C/LAM?

- Collaborative Application Lifecycle Management
  - Alineando Requerimientos, Desarrollo y Pruebas
- C/LAM agiliza la entrega de software debido a:
  - Coordinar las actividades del ciclo de vida de desarrollo de software
  - Permitir la trazabilidad a través de artefactos
  - Conocer el status del proyecto en tiempo real
  - Permite la integración de productos RTC, RRC y RQM
- Basado en Jazz

5.1.5. Lo Novedoso de Jazz

Si se quiere colaborar con analistas empresariales, arquitectos, desarrolladores, probadores, abogados, accionistas empresariales y otros expertos que se encuentran en otro lugar o en otra organización. ¿Qué tipo de infraestructura permitiría desempeñar mejor el trabajo, tanto de forma individual como en equipo? La plataforma de tecnología Jazz está diseñada para

responder a estas necesidades. Los productos basados en la plataforma Jazz permitirán a los miembros del equipo:

- *Colaborar en contexto:* La tecnología Jazz realiza un seguimiento de las relaciones entre artefactos y las gestiona, fomenta procesos sólidos de desarrollo y recopila información del proyecto automáticamente y de forma discreta para proporcionar una integración del ciclo de vida.
- *Disponer del nivel adecuado de gobierno:* La plataforma Jazz permite a los equipos detectar, compartir y automatizar métodos recomendados con diferentes grados de rigor. Resulta muy sencillo adaptar el proceso de gobierno a lo largo del tiempo y evitar así tener que definir todos los elementos desde el principio. Así mismo, se puede variar la flexibilidad del proceso mientras dure el proyecto para facilitar la experimentación en las etapas iniciales y la estabilización en las finales. La comprensión del proceso y la automatización incorporada permiten a los individuos y a los equipos "hacer lo adecuado" y "de forma adecuada".
- *Ofrecer productividad desde el primer día:* La plataforma Jazz ha sido diseñada para hacer posible la configuración directa y el suministro dinámico de nuevos proyectos y miembros de equipos. Una vez que los equipos y los proyectos se han configurado, se pueden planificar iteraciones fácilmente, equilibrar cargas de trabajo, y modificar de forma dinámica el trabajo colectivo de los equipos. Al agilizar la incorporación, las empresas obtienen la flexibilidad necesaria para responder de forma más rápida a las necesidades de la organización y aprovechar los conocimientos técnicos y empresariales estén donde estén.
- *Elegir su propio método:* La tecnología Jazz está basada en los estándares abiertos WEB y OSGi. Su arquitectura abierta y ampliada está diseñada para otorgar la flexibilidad de montar una plataforma de software propia, teniendo en cuenta los proveedores y las soluciones que se prefieran. La arquitectura de Jazz permite utilizar múltiples clientes. Jazz cuenta con una interfaz abierta de middleware, lo que significa que se podrán instalar productos basados en Jazz mediante middleware de código abierto, como Tomcat, Derby y Jabber; middleware comercial de IBM WebSphere, DB2 y Lotus, e incluso productos de terceros, como Oracle.

5.1.6. Características y servicios proporcionados por Jazz Team Server

Entre las características y los servicios comunes proporcionados por Jazz Team Server podemos identificar los siguientes:

- *Utilización de una plataforma estándar para la integración de las herramientas con un repositorio único de información (servidor de aplicaciones y base de datos estándares).*
- *Control de acceso unificado:* Gestión de usuarios centralizada e integrada con sistemas LDAP. Personalización de permisos en Jazz para permisos específicos de cada herramienta.
- *Nomenclatura y conceptos unificados para todas las herramientas de apoyo al ciclo de vida de desarrollo de software (concepto único de proyecto de desarrollo y posible particularización para definir diferentes equipos que trabaje en el mismo proyecto).*
- *Acceso remoto a través de una interfaz Web (tráfico http/https). Ver figura 5.5.*
- *Interfaz Web intuitiva desarrollada con las tecnologías Web 2.0.*
- *Integración en el entorno de desarrollo Eclipse.*
- *Integración con herramientas de colaboración email, sistemas de mensajería instantánea.*

- *Definición, adaptación y modificación de procesos de desarrollo para todo el ciclo de vida desde un sitio centralizado.* Posibilidad de adaptar ese proceso a cada línea de desarrollo y definir reglas específicas de validación.
- Consola centralizada desde la cual se puede consultar el avance y estado del desarrollo en tiempo real.
- *Notificación automática de eventos*, registrando de manera transparente y automática los eventos que se van produciendo durante el desarrollo.
- *Plataforma abierta a otras integraciones con servicios de colaboración comunes.*

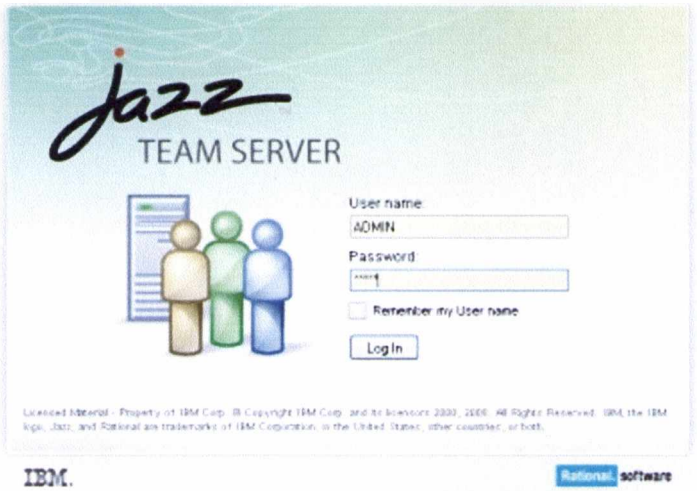

Figura 5.5. Interfaz Web Jazz

5.1.7. Ventajas

- *Mayor visibilidad y transparencia en el desarrollo, reduciendo la sobrecarga de informes.*
- *Mayor previsibilidad y menor riesgo en el desarrollo gracias a la consola centralizada con información en tiempo real de todas las etapas del ciclo de vida de desarrollo.*
- *Entorno adaptado al trabajo con factorías o empresas externas.*
- *Ahorro de costes de instalación e integración de herramientas.*
- *Ahorra tiempo en formación a usuarios finales*
- *Mejora de comunicación entre los diferentes miembros del equipo de desarrollo.*
- *Uso de estándares de mercado. Desarrollo transparente y abierto a la comunidad.*
- *Productividad desde el primer día.*
- *Integración dinámica de personas, procesos y proyectos.*

5.2. Rational Team Concert

El software IBM Rational Team Concert (Figura 5.6.) es un entorno de colaboración que intenta ayudar a personas y equipos a rendir al máximo. Proporciona control esencial de versiones, gestión del espacio de trabajo y capacidad de desarrollo en paralelo. Además, Rational Team Concert está diseñado para conectar equipos de desarrollo dispersos para incrementar la productividad individual y del equipo, comprimir los ciclos de desarrollo y permitir el suministro de software de alta calidad en poco tiempo. Disponible en ediciones Standard, Express y Express-C (Y la más novedosa Enteprise), Rational Team Concert ha sido creado específicamente para equipos de desarrollo pequeños y medianos.

- **IBM Rational Team Concert (Conceptos claves)**
  - Transparente
  - Presencia integrada
  - Wikis
  - Abierto
  - Chat
  - Reportes en tiempo real
  - Automatización libre de intervención del usuario
  - Web 2.0
  - Indicadores de desempeño personalizados
  - Recolección de datos automatizado
  - Extensibilidad
  - Plug-Ins de Eclipse
  - Arquitectura de servicios
  - Libertad para crear

Figura 5.6. Rational Team Concert

5.2.1. Motivación

Las organizaciones de desarrollo de software están sometidas a crecientes presiones para suministrar rápidamente software de alta calidad coherente con los objetivos empresariales, que cambian constantemente. Sin embargo, esta meta resulta cada vez más difícil de alcanzar debido a varios factores. Se espera con frecuencia que los equipos creen más aplicaciones con los mismos o menos recursos; muchos equipos están dispersos geográficamente, lo que complica la colaboración, y numerosas organizaciones encuentran dificultades para aplicar procesos y estándares coherentes en proyectos y grupos dispares.

La colaboración nunca ha sido tan importante para los desarrolladores de software (ni tan difícil).

En primer lugar, como el software se vuelve más complejo y las expectativas de los usuarios aumentan, el desarrollo de software requiere de un enfoque orientado al equipo en lugar de esfuerzos individuales.

En segundo lugar, las metodologías modernas, como el desarrollo ágil requieren cada vez más estrecha colaboración entre los miembros del equipo para lograr una velocidad que no permita esperar hasta la próxima reunión para identificar los temas en cuestión.

En tercer lugar, el ciclo de desarrollo es cada vez más comprimido, ya que las empresas se esfuerzan por conseguir la mayor ventaja competitiva en desarrollo de sus software.

Por último, la colaboración no significa sólo trabajar con alguien en el mismo lugar físico. Ahora también se deben integrar a personas que se encuentran en cualquier parte del mundo.

La forma tradicional de colaborar en un proyecto de desarrollo de software es tener reuniones (muchas reuniones). Pero esa estrategia es una limitación porque los equipos en

las reuniones no están produciendo resultados. Lo que quizá sea peor, la colaboración en estas reuniones tiende a conducir a una mentalidad de "cascada", donde los problemas que afectan al equipo no se resuelven, hasta después de la próxima reunión.

El desarrollo de software en equipo debe ser parecido a formar parte de una banda de música: en ambos casos se necesita alcanzar un equilibro entre habilidad y colaboración.

Como respuesta al interrogante de la colaboración, IBM introduce Jazz para equipos de desarrollo de software. Jazz es una nueva plataforma de desarrollo colaborativo de IBM, diseñada para ser la base de aplicaciones que permiten que equipos cada vez más dispersos, trabajen más de cerca. Jazz no es una aplicación, sino más bien una plataforma abierta en la que IBM y otros pueden construir aplicaciones para satisfacer las necesidades de un nuevo mundo colaborativo.

Rational Team Concert es la principal aplicación que lanzó IBM para este propósito.

Rational Team Concert está diseñado en torno a una constante: continua colaboración donde un equipo está siempre en contacto a través de la Web... todos los miembros del equipo están al corriente de todo (reuniones, comunicaciones telefónicas e informes formales se sustituyen por la comunicación ad hoc sobre Internet).

Con Rational Team Concert cada miembro del equipo puede saber las actividades de los otros miembros del equipo y la relación de esas actividades con las propias. Tal vez lo más importante de todas las interacciones entre los miembros del equipo es que se tratan como activas, se almacenan y se relacionan con la parte del proyecto a la que refieren.

Rational Team Concert es lo suficientemente flexible ya que se apoya en una amplia variedad de modelos de procesos, dependiendo del tamaño del equipo, la complejidad del proyecto y la cultura del desarrollo. Si un proyecto necesita un plan de desarrollo muy estructurado, Team Concert lo soportará. Si un equipo pequeño necesita rapidez, agilidad y un enfoque poco estructurado, Team Concert también lo soportará.

La colaboración en el desarrollo de software no es un lujo sino una necesidad en la que los equipos de software tiene que averiguar cómo hacerla mejor. Con las herramientas adecuadas, tales como IBM Rational Team Concert (Construida sobre la tecnología de Jazz), los equipos de desarrollo de software pueden sincronizarse en tiempo real, independientemente de su ubicación.

5.2.2. Fundamentos

Tanto la tecnología Jazz como RTC se basan en 3 conceptos fundamentales, para el desarrollo de software colaborativo, que mejoran el conocimiento y la madurez de las prácticas dentro de un ambiente que incrementa el talento individual y del equipo.

- *Colaboración:* Transparencia de los equipos y proyectos para la colaboración continua y sensible al contexto.
- *Automatización:* Automatizar tareas no creativas con procesos automatizados y workflows.
- *Reportes:* Entregar información al instante acerca de programas, proyectos y utilización de recursos.

5.2.2.1. Por que colaborar?

- Para acelerar la eficiencia del proyecto y del equipo
- Para crear cohesión en el Equipo
- Para fomentar el talento a lo largo y más allá de la empresa
- Para flexibilizar modelos de uso de recursos

El resultado será, acelerar el tiempo de salida del software

5.2.2.2. Por que automatizar?

- Para facilitar el "Hacer lo correcto"
- Para reducir las tareas manuales, tediosas y que consumen tiempo
- Para minimizar los tiempos de espera improductivos
- Para minimizar la carga administrativa

El resultado será, mejorar la calidad del software

5.2.2.3. Por que reportar?

- Identificar y resolver los problemas más temprano en el ciclo de software
- Obtener métricas basadas en hechos, no en estimados, para mejorar la toma de decisiones
- Fomentar el uso de métricas para la mejora individual o del equipo.

El resultado será la reducción del costo

5.2.3. Básicamente, Que es Rational Team Concert?

Un novedoso producto basado sobre la tecnología Jazz optimizado para equipos pequeños y medianos de desarrollo que integra el equipo completo alrededor de un servidor de integración y que incluye un ambiente de desarrollo mejorado en Eclipse (Figura 5.7). También esta la versión cliente en Visual Estudio. Incluye una interfaz Web Cliente (Figura 5.8.). Esta disponible en versiones Express-C, Express, Standard (y la novedosa Enterprise).

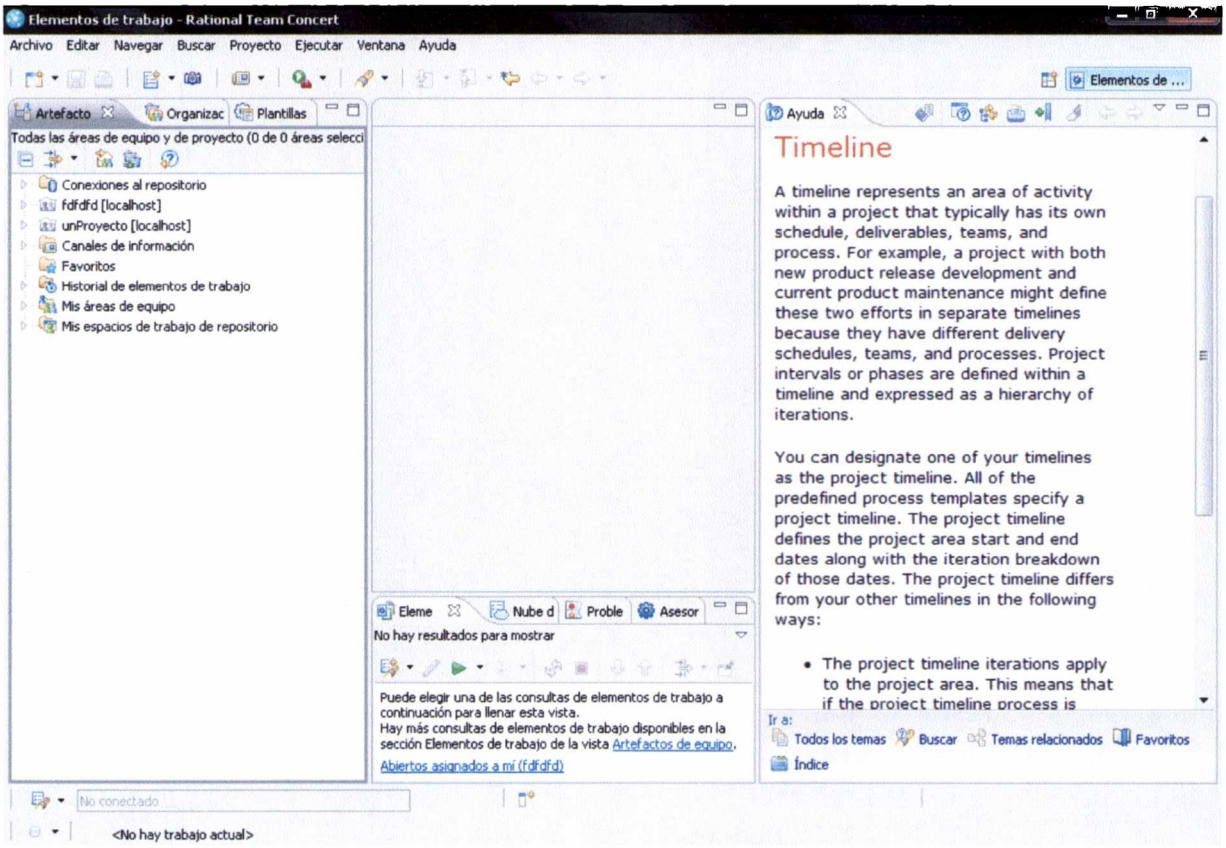

Figura 5.7. Área de trabajo de RTC Eclipse

Las características principales de RTC son:

- Ambiente colaborativo entre los miembros del equipo
- Soporte y ejecución para procesos de desarrollo

- Transparencia de status y tendencias a través de reportes en tiempo real que muestran el estado del Proyecto y permiten un rápido análisis de datos
- Uso de Dashboard personalizados

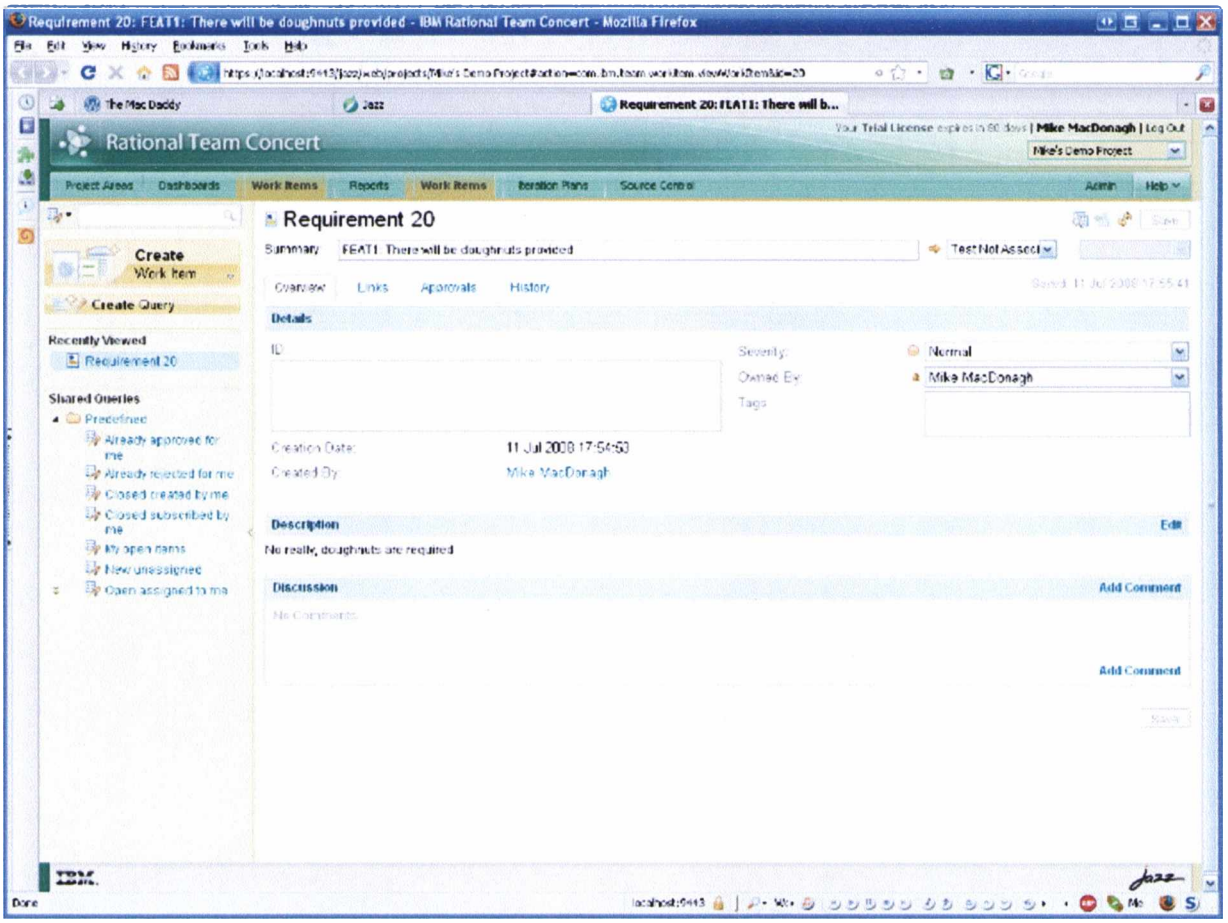

Figura 5.8. Interfaz Web de RTC

Innovación en Software a través de la Colaboración:

- **Colaboración en tiempo real y en contexto de equipos de trabajo**
  - Hace al desarrollo de software más automatizado, transparente y predictivo
- **Pensar y Trabajar al Unísono**
  - Control de fuentes, Artículos de Trabajo (Work Items) y administración de builds integrados
- **Evaluar el estado de Proyecto en tiempo real**
  - Captura datos automáticamente
- **Automatizar mejores prácticas**
  - Los procesos dinámicos aceleran el flujo de trabajo del equipo
  - Procesos personalizados ó Out-of-the-box
- **Unificar equipos de trabajo de software**
  - Integración de un gran conjunto de herramientas y clientes
  - Extensión del valor de ClearQuest y ClearCase (Ambos, productos de IBM)
  - Clientes Visual Studio y Eclipse
  - Soporte a servidores System z y System i
- **Abierto y extensible en Jazz**
  - Colabore en Contexto
  - Gobernabilidad a la Medida

En la figura 5.9. se contrastan las características de las diferentes ediciones de RTC.

	Express-C para consultores y estudiantes	Express para departamentos/ pequeñas y medianas empresas	Standard para equipos corporativos
Número máximo de desarrolladores/contribuidores por servidor	10 (total)	50/ilimitado	250/ilimitado
Base de datos incluida (opcional)	Solo Derby	IBM DB2 Express (DB2, Oracle, Microsoft® SQL Server)	IBM DB2 Express (DB2, Oracle, Microsoft SQL Server)
Servidor de aplicaciones incluido (opcional)	Solo Apache Tomcat	Tomcat (IBM WebSphere)	Tomcat (IBM WebSphere)
Gestión de código fuente	•	•	•
Seguimiento de elementos de trabajo	•	•	•
Gestión de compilaciones	•	•	•
Planificación ágil	•	•	•
Integración en Subversion	•	•	•
Permisos a nivel de servidor	•	•	•
Autentificación Lightweight Directory Access Protocol (LDAP)	•	•	•
Proceso personalizable	•	•	•
Paneles de control	1 por proyecto	1 por proyecto	Ilimitados
Informes			•
Atributos y flujo de elementos de trabajo personalizables			•
Permisos de proceso por roles			•
Conector Rational ClearCase			•
Conector Rational ClearQuest			•
Importación LDAP			•
Soporte proxy HTTP estándar			•

Figura 5.9. Tabla comparativa entre ediciones de RTC

5.2.4. Beneficios y características generales de RTC

Aspectos destacados de RTC

- Agiliza los procesos de desarrollo con capacidades de control de fuentes, elementos de trabajo y compilaciones
- Mejora la colaboración mediante colaboración contextual en tiempo real para equipos de proyecto distribuidos
- Ayuda a mejorar la calidad del software impulsando procesos consistentes
- Mejora la visibilidad del proyecto con información del estado en tiempo real y transparencia
- Incrementa la flexibilidad mediante una plataforma basada en estándares abiertos
- Mejora la productividad mediante control esencial de versiones, gestión del espacio de trabajo y capacidad de desarrollo en paralelo

- **Mejora la colaboración en el equipo**

Rational Team Concert combina la metodología Web 2.0 para la colaboración en equipo con la gestión integrada de elementos de trabajo, compilaciones y configuración del software. Mediante vistas de portal personalizadas, los miembros del equipo pueden acceder a información sobre los proyectos, como noticias e incidencias, estado actual de la compilación, elementos en los que se está trabajando y cambios solicitados. También pueden ver en qué están trabajando sus compañeros de equipo, así como saber quién está conectado y disponible para colaborar. Con Rational Team Concert es fácil intercambiar información directamente en el contexto del trabajo realizado. Rational Team Concert puede, además, poner al día automáticamente a los participantes de cualquier cambio en el documento o elemento de trabajo asociado.

- **Permite la gestión transparente de elementos de trabajo**

  Rational Team Concert crea y sigue la evolución de cada elemento de trabajo de acuerdo con el proceso del equipo y las normas del proyecto. Esta función permite el flujo efectivo de defectos, mejoras y conversaciones entre todo el equipo, acelerando el progreso del proyecto. Rational Team Concert captura información sobre el avance de los trabajos, como quién, qué, cuándo y por qué, y la asocia a cada elemento de trabajo para proporcionar el contexto indispensable a la hora de compartirlos entre los miembros del equipo. Las funciones de aprobación y debate ayudan a garantizar que las evaluaciones y aprobaciones de código tienen lugar según lo previsto y que los miembros del equipo permanecen coordinados.

- **Proporciona amplias capacidades de gestión de control de fuentes**

  Rational Team Concert proporciona control esencial de versiones, gestión del espacio de trabajo y capacidad de desarrollo en paralelo para personas y equipos. Puede ayudar a mejorar la productividad siguiendo automáticamente los cambios realizados en los artefactos y permitiendo retroceder a cualquier versión anterior. Con Rational Team Concert, los equipos pueden alcanzar el equilibrio adecuado entre experimentación y transparencia mediante combinaciones de espacios de trabajo privados y públicos. La gestión integrada del flujo y "sandboxes" basados en servidor ayudan a los miembros del equipo a resolver problemas en un entorno controlado y contribuir así fácilmente a las compilaciones regulares del equipo.

- **Orientación automática durante el proceso**

  Rational Team Concert ofrece funciones de automatización y orientación durante el proceso con las que puede controlar el proceso exactamente en la medida necesaria como para garantizar resultados predecibles. Debido a que no todos los proyectos y organizaciones son iguales, Rational Team Concert permite, además, configurar normas para satisfacer necesidades concretas del proyecto o la organización. Puede comenzar con cualquiera de los tres modelos de proceso listos para usar que ofrece y posteriormente definir y modificar las normas sobre la marcha. Con Rational Team Concert los equipos pueden adoptar los principios básicos de los equipos de software de alto rendimiento: ciclos de iteración más breves, participación habitual del cliente y compilaciones más frecuentes y automatizadas. La orientación que proporciona al proceso, completamente personalizable, automatiza la implementación de casi cualquier proceso de suministro de software y es válido tanto para equipos ágiles como tradicionales.

- **Permite crear compilaciones de software más eficaces**

  Rational Team Concert proporciona amplias funciones de gestión de compilaciones que pueden ayudar a su equipo a programar y ejecutar compilaciones de software eficazmente. La capacidad de seguimiento integrada de elementos de trabajo y conjuntos de cambios facilita la identificación exacta del punto en el que la compilación falla y sus detallados informes registran automáticamente las actividades del equipo. La integración constante de compilaciones permite tanto a equipos ágiles como tradicionales beneficiarse de la mayor productividad asociadas a los procesos de compilación continuos.

- **Mejora la visibilidad con información en tiempo real del estado del proyecto**

  Rational Team Concert dispone de funciones automáticas de recopilación de datos y elaboración de informes con las que dispondrá del conocimiento en tiempo real necesario para dirigir eficazmente sus proyectos de software, reduciendo simultáneamente el tiempo dedicado a estas tareas. De esta forma será más fácil mantenerse al día del progreso del proyecto, resolver posibles problemas y emprender actuaciones correctivas al principio del ciclo de vida de éste.

- **Ayuda a ampliar sus inversiones en la empresa**

  Rational Team Concert puede integrarse fácilmente en otras aplicaciones de gestión de cambios. Es posible integrar el software de Rational con el software de control de versiones Subversion. Utilizando conectores, se puede integrar el software RTC en las aplicaciones IBM Rational ClearCase e IBM Rational ClearQuest. Dichos conectores permiten sincronizar datos entre RTC y el software Rational ClearCase o Rational ClearQuest, ayudando a los equipos a utilizar los activos y procesos de la empresa. Además, con Rational Team Concert, los desarrolladores pueden transferir el trabajo de los equipos satélite directamente a los proyectos de la empresa.

- **Cuenta con las ventajas del ecosistema de IBM Business Partners**

  Puede ampliar Rational Team Concert con soluciones Ready for IBM Rational creadas por IBM Business Partners que proporcionan capacidades especializadas y funciones de valor añadido. Por ejemplo, los IBM Business Partners pueden ofrecer funciones como:

  - Análisis e informes sobre licencias de software y conformidad del código fuente.
  - Mecanismos de proceso para automatizar de forma transparente la dirección interna de la arquitectura orientada a servicios (SOA) y otras políticas.
  - Transformación automática de texto y validación de requisitos.
  - Integración bidireccional de Rational Team Concert y la gestión de cambios JIRA.
  - Dirección interna de desarrollo ligera para equipos ágiles y tradicionales.

- **Utiliza la plataforma tecnológica Jazz**

  Rational Team Concert es la primera aplicación basada en la tecnología Jazz de IBM, la plataforma tecnológica de nueva generación para suministro de software de colaboración de IBM. Creada a partir de estándares Open Web y de la OSGi Alliance, la plataforma Jazz proporciona una arquitectura ampliable diseñada para mejorar la colaboración, productividad y transparencia en el desarrollo de software. Adaptada a las necesidades de los equipos mundiales, combina la información de personas, proyectos y procesos con la automatización para acelerar el ciclo de vida del software y mejorar la dirección del proyecto.

En la figura 5.10. se pueden observar gráficamente características de RTC.

Figura 5.10. Funcionalidades principales de RTC

5.2.5. Para qué utilizar Rational Team Concert?

¿Cuáles herramientas conocen

- ... acerca de tu equipo?
- ... acerca de todos tus artefactos? (Ej. RPG, EGL, solicitudes de cambio, planes de iteración)
- ... quien es responsable de que?
- ... reglas bajo las cuales será liberado el código? (Calidad del Código, Trazabilidad, Pruebas de Ejecución, Propiedad Intelectual)
- ... cómo configurar un nuevo proyecto?
- ... cómo ayudar a un nuevo miembro del equipo a iniciar el desarrollo?
- ... el proceso de desarrollo, el favorito tipo de work item y el workflow asociado al mismo
- ... cuando el build funciona y qué hacer si este falla?

Se puede utilizar RTC para:

- Desarrollar procesos predefinidos con iteraciones, roles de usuario y controles de acceso.
- Personalizar procesos para configurar reglas (por ejemplo, no permitir que cualquier usuario haga check in de códigos fuente Java que tienen errores de compilación), roles de usuario y tipos de work item para una organización. RTC es muy flexible.
- Crear planes para release del proyecto y cualquier iteración (milestones) dentro del tiempo de entrega del build
- Crear gráficos para desplegar el status del proyecto (por ejemplo, cuántos defectos abiertos vs. cerrados)
- Realizar control de versión del código escrito en RPG, COBOL, Java, EGL.
- Seguimiento de cambio de las aplicaciones a través de work items (tareas, defectos, etc.)
- Auditar un artefacto en particular (¿quién hace qué cambios? ¿Cuando y dónde fueron hechos los cambios? ¿Por qué y donde fueron hechos los cambios?)
- Compilar los artefactos de aplicación incluyendo RPG, COBOL, DDS, CL y fuentes de Java, on demand o planificado.

5.2.6. Rational Team Concert desde adentro. Características.

En los próximos apartados se desarrollaran y enumeraran diversas características del software RTC, así como también, se presentarán diferentes screen-shots autoexplicativos.

5.2.6.1. Desarrollo Globalmente Distribuido

- Rational Team Concert esta disponible en sus versiones Standard, Express, Express-C, y Enterprise
- Es un entorno de desarrollo independiente optimizado para equipos pequeños y medianos
- Sus versiones son escalables para el desarrollo globalmente distribuido incluso a nivel mundial (Figura 5.11.)
- Explota todas las capacidades de colaboración de la plataforma de Jazz, además de los Works ítems integrados, gestión de SCM y Builds
- Dispone de Dashboards y Reportes en tiempo real
- Favorece la conciencia de equipo y proceso

Figura 5.11. Versiones son escalables para el desarrollo globalmente distribuido

5.2.6.2. Unificar los equipos distribuidos

- Permite desarrollar proyectos "satélites" o ágiles opcionalmente integrados con productos de infraestructura en Rational ALM
- Aumenta la oferta ya existente de ALM con nuevas capacidades de proceso y colaboración

En la figura 5.12. se muestra gráficamente esta integración.

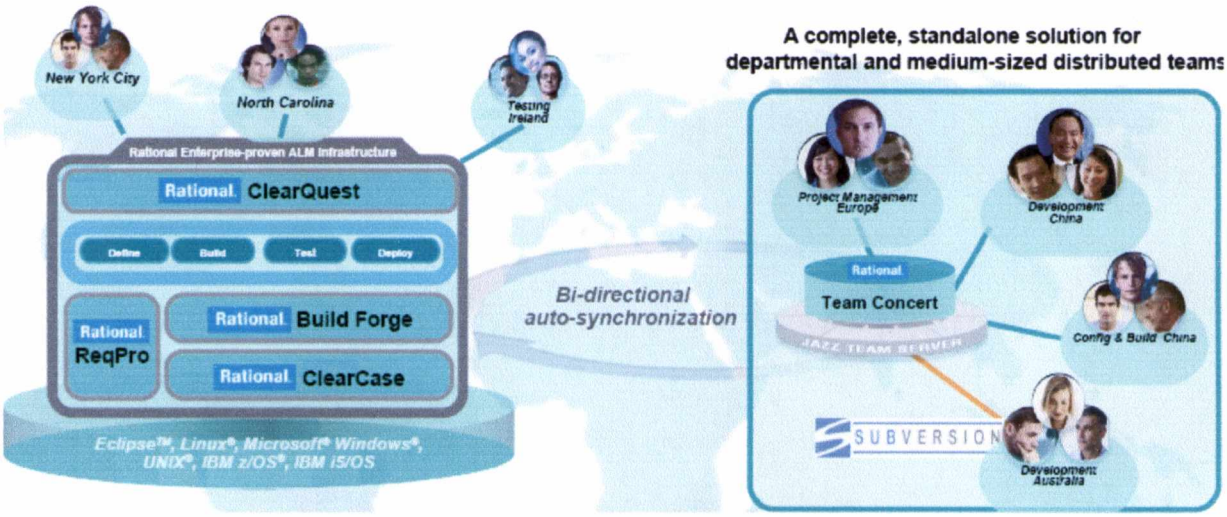

Figura 5.12. Integración y sincronización de productos ALM

5.2.6.3. Facilita el conocimiento del proceso y guía

- Tiene incorporado un "Team Advisor" que asesora al equipo (Figura 5.13.)
- Se pueden establecer reglas cuando se entregan los cambios para forzar los estándares de equipo o los estándares organizacionales.
- Ayuda en asegurar resultados de mayor calidad al forzar el uso de estándares establecidos.
- Las reglas son configurables
- Se pueden especificar "quick fixes" para simplificar acciones correctivas
- Las reglas de procesos se pueden definir, refinar "al vuelo", habilitando mejoras continuas.
- Procesos predefinidos disponibles, incluyendo: OpenUp, Scrum, The Eclipse Way, Agile (XP-like), etc.

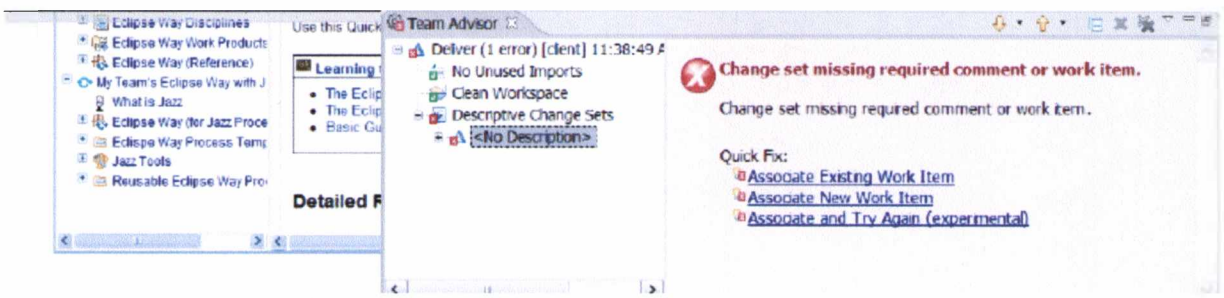

Figura 5.13. Team Advisor

5.2.6.4. Procesos Ágiles incluidos con Rational Team Concert

- Plantilla de Proceso Ágil: La plantilla de proceso Ágil provee un mash-up de mejores prácticas de varios procesos ágiles existentes. Incorpora el estilo XP de métodos ágiles.

- La plantilla Eclipse Way: El proceso Eclipse Way es un proceso ágil, basado en iteraciones con un foco en la entrega consistente a tiempo y con calidad del software.
- La plantilla de proceso OpenUp: El proceso OpenUp mantiene las características esenciales de Rational Unified Process, el cuál incluye desarrollo iterativo, casos de uso y escenarios que determinan el desarrollo, la administración de riesgos y el método centrado en arquitectura.
- Plantilla de proceso ejemplo de Cloudburst: La plantilla de proceso ejemplo de Cloudburst está diseñada para soportar el proyecto de referencia de Cloudburst, el cuál está disponible con la plataforma de tecnología jazz.
- Plantilla de proceso Scrum: La plantilla de proceso Scrum soporta un método popular para administrar los proyectos en la forma ágil.
- Plantilla de proceso Simple Team: La plantilla de proceso Simple Team le permite a los equipos iniciar rápidamente. Los miembros del equipo tienen permisos para realizar cualquier modificación en el proyecto.

5.2.6.5. Consideraciones básicas para integrarse a un nuevo equipo

- *Unirse a un equipo*
  - Para la mayoría de los entornos, unirse a un proyecto puede ser complicado. Team Concert lo intenta hacer lo mas fácil posible.
  - Agregar un nuevo miembro de equipo a un proyecto genera una invitación por correo electrónico
  - El contenido del correo electrónico puede ser utilizado para configurar el acceso del nuevo miembro del equipo a los recursos del proyecto en Team Concert
- *Comunicación*
  - Los usuarios de Team Concert puede utilizar una variedad de herramientas para comunicarse con el resto del equipo
    - E-mail
    - Mensajería instantánea / Chat
    - RSS feeds
    - Web UI
    - Cliente Team Concert
  - Los miembros pueden utilizar todos los mecanismos de comunicación típica para trabajar juntos como un equipo, independientemente de dónde se encuentren físicamente. Esta colaboración permite una vista única de los datos del proyecto
    - Mensajería instantánea integrada para feedback inmediato
    - RSS feeds para avisar sobre eventos importantes en el proyecto en tiempo real
    - Interfaz de usuario Web utilizada por cualquier persona en el equipo, o por quien tenga interés en el proyecto

5.2.6.6. Software Configuration Management

Software Configuration Management (SCM) es una especialización de la Gestión de configuración a todas las actividades en el sector del desarrollo de software.

SCM trata y controla:

- La elaboración de código fuente por varios desarrolladores simultáneamente,
- El seguimiento del estado de las versiones y sus cambios

- La conducción de la integración de las partes del software en un solo producto de software.

A continuación se especifican puntos relativos a RTC.

5.2.6.6.1. Unidades de Trabajo (Work Items)

- Rational Team Concert permite dividir el trabajo en Work Items.
- El conjunto de tipos de Work Items es abierto
- Los tipos estándar son: Tarea, mejora, defecto
- El conjunto es definido por cada equipo
- El ciclo de vida de cada Work Item es configurable
- Todos los Work Items son almacenados en un repositorio
- Se permiten consultas sobre los Work ítems
  - RTC proporciona buena información del proyecto en tiempo real y transparencia del estado mediante la recopilación de datos automatizado.
  - Rational Team Concert proporciona un mecanismo de consulta para la búsqueda de Work Items en un área de proyecto que permite una mayor transparencia del proyecto.
    - El ámbito de consulta para los Work Items es el área del proyecto.
  - La interfaz de usuario incluye:
    - Un editor para la creación de consultas estructuradas de Work Items
    - Una vista de Work Items configurable de usuario final para explorar los resultados de la consulta.
- La forma en que Rational Team Concert utiliza los Works Items significa que todo el trabajo puede ser planificado, seguido y almacenado bajo la gestión de configuración. De esta manera se permiten vistas del estado de los datos y del proyecto. En la figura 5.14. se muestra y describe la forma en la que RTC administra los Work Items.

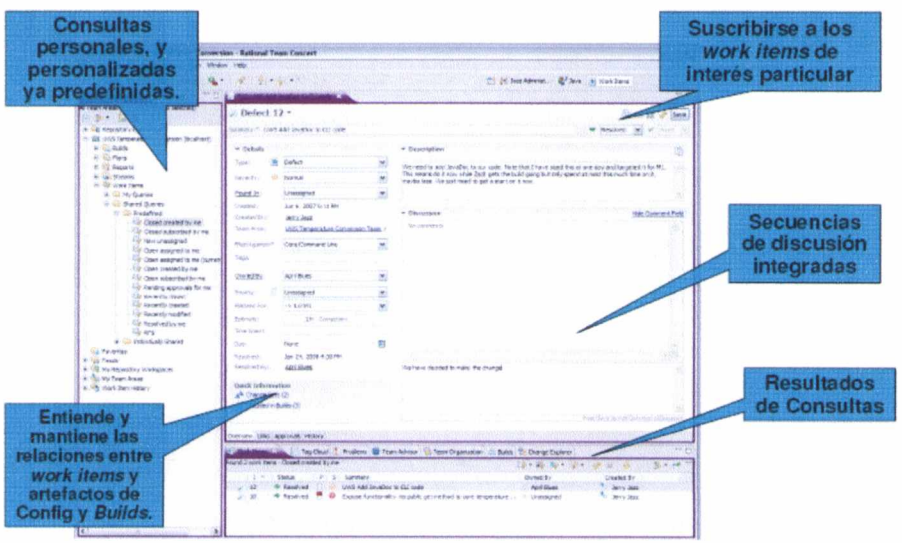

Figura 5.14. Administración de Work Items en RTC

- Los Works Items son muy importantes en Rational team Concert (Figura 5.15)

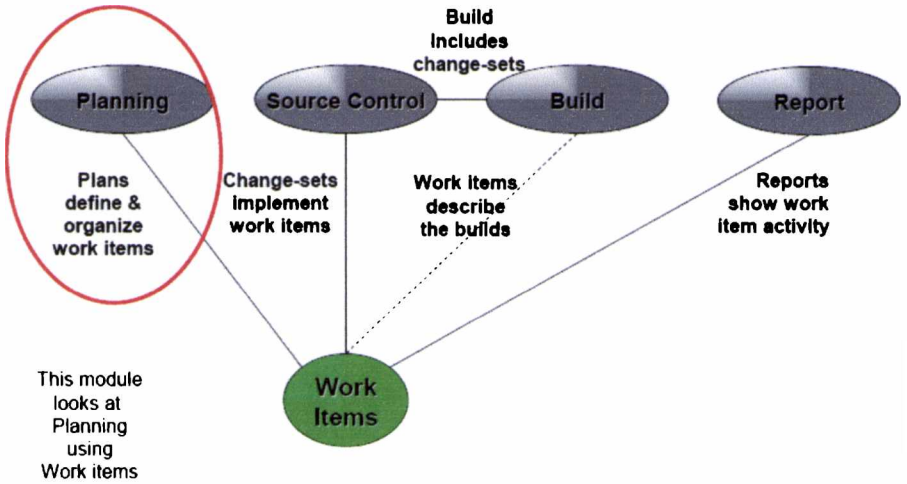

Figura 5.15. Importancia de los Work Items

5.2.6.6.2. Planificación de iteraciones

- Un plan es una colección de Work Items...
  - Asignado a un hito determinado
  - Para un equipo determinado
- Modificaciones en los planes
  - Si cambian los Work Items cambia el plan
  - Si se cambia el plan directamente, cambiar los Works Items
  - Crear nuevos Work Items a partir del plan
- La estructura del plan es dinámica
  - Fácilmente se agruparán por propietario, categoría, duración, prioridad, etc. En la figura 5.16. se muestra y explica gráficamente como RTC administra las iteraciones.

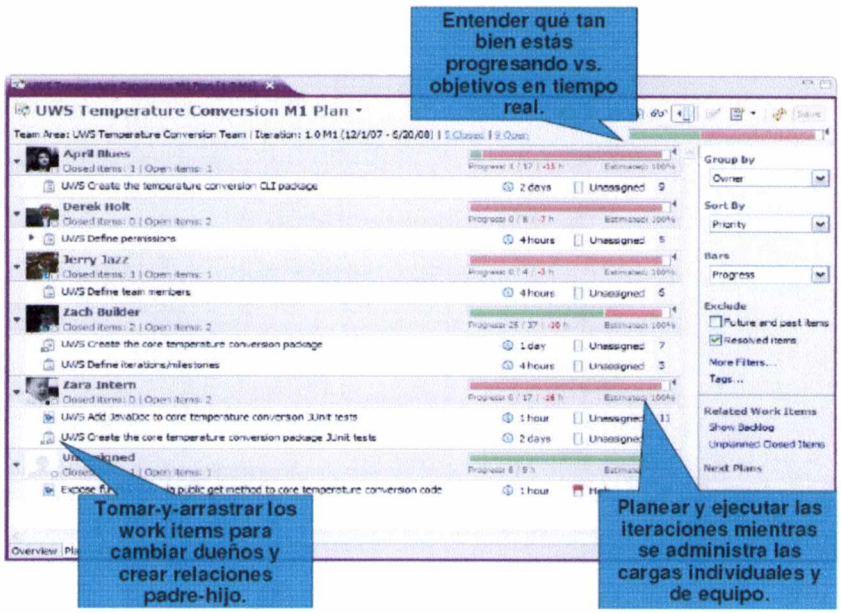

Figura 5.16. Administración de iteraciones en RTC

- Los planes son visibles
  - Disponibles para todo el equipo
  - Observables por interesados

En la figura 5.17. se muestra gráficamente los pasos por los que pasan las iteraciones en RTC.

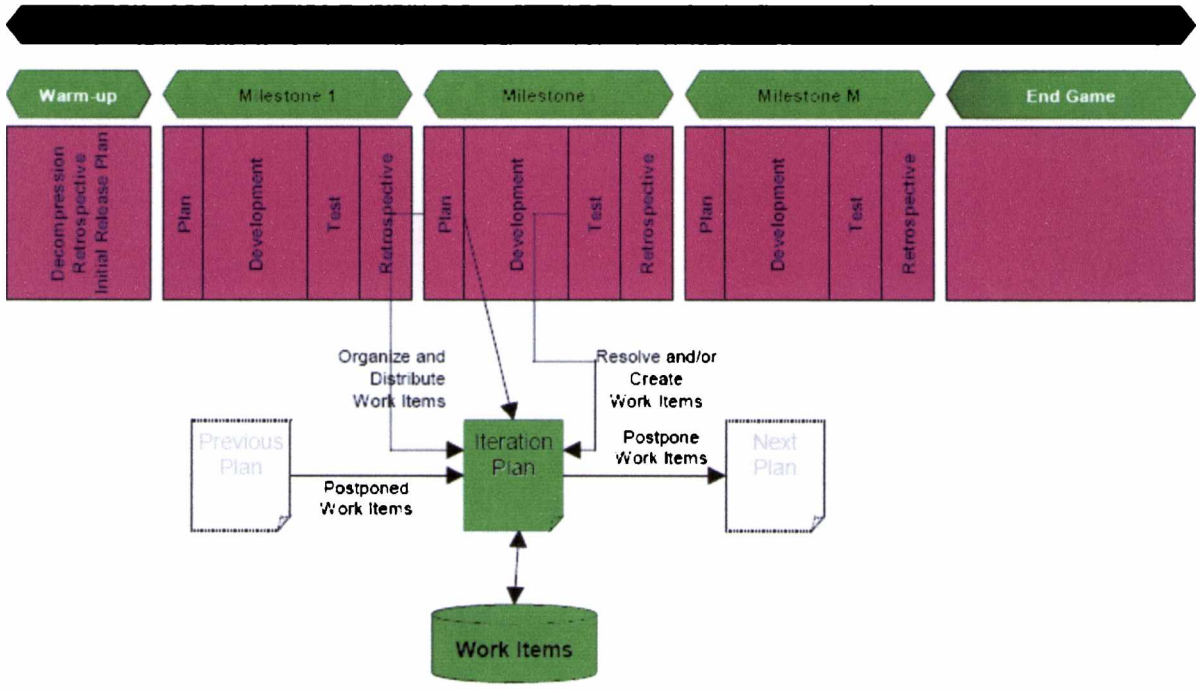

Figura 5.17. Pasos de las iteraciones

5.2.6.6.3. Colaboración en tiempo real

- La capacidad de generar consultas y generar reportes están disponibles en clientes Eclipse y en interfaz de usuario Web (Figura 5.18.).
  - Definir consultas sobre Work Items para encontrar el trabajo propio y el de otros.
  - Ver quién está conectado y dispuesto a colaborar.
  - Ver el registro de eventos que son interesantes y seguir las noticias vía RSS.
  - Generar, visualizar y exportar informes sobre el estado y la evolución del proyecto.

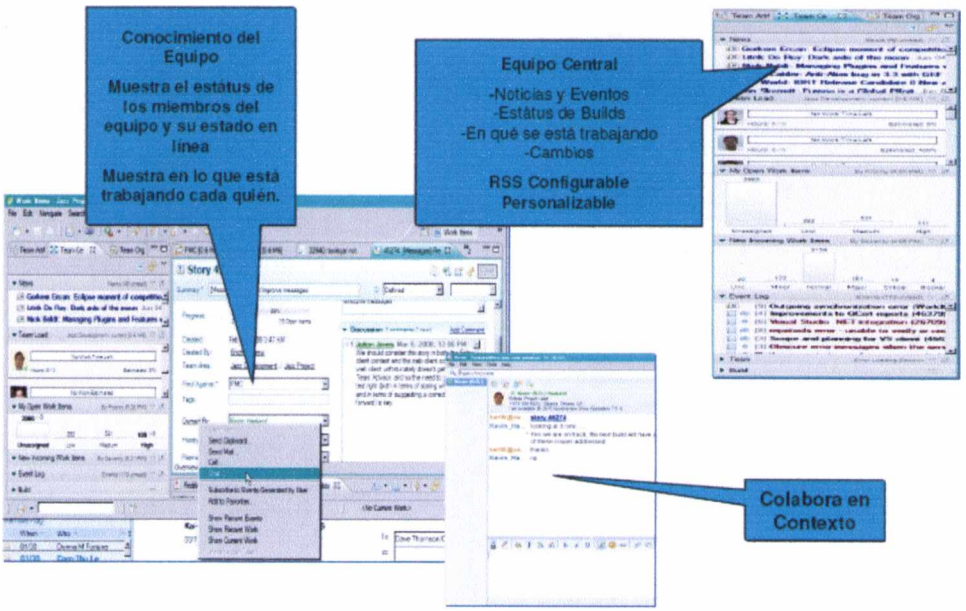

Figura 5.18. Colaboración en tiempo real

- Rational Team Concert muestra la información mediante vistas actualizadas automáticamente que son configurables, de forma tal que la información necesaria, se obtiene en tiempo real
- *Depuración Colaborativa:* Se pueden compartir sesiones para hacer debugging, utilizando Rational Team Concert y Rational Application Developer, como lo muestra la figura 5.19.

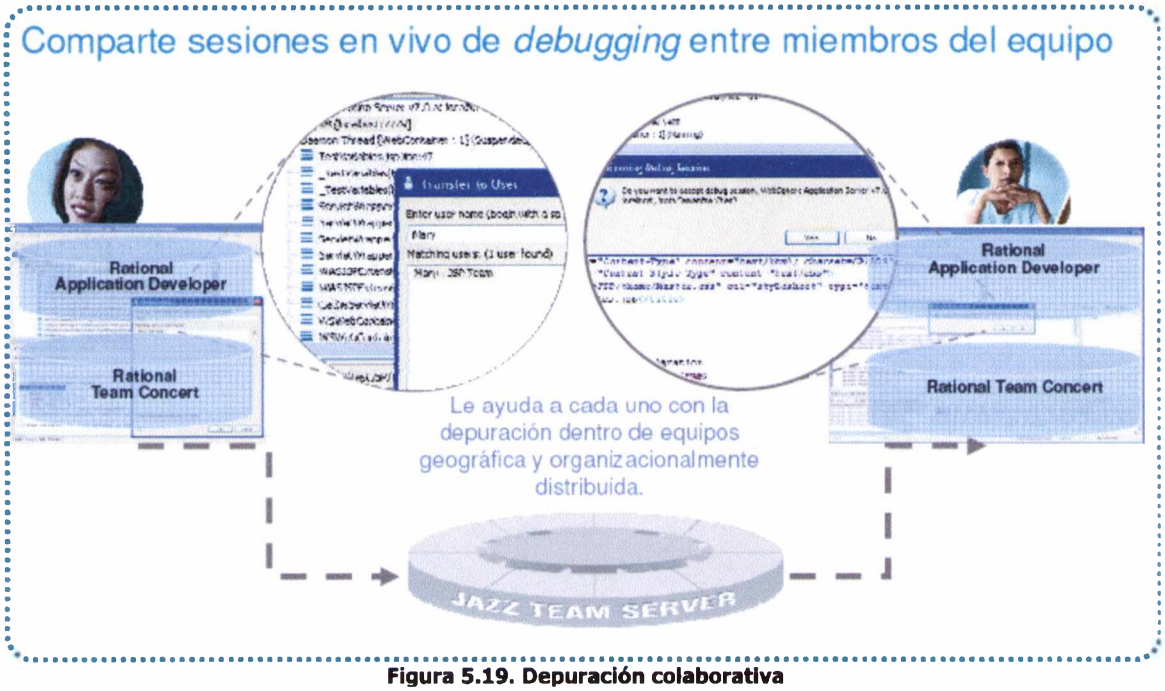

Figura 5.19. Depuración colaborativa

5.2.6.6.4. Administración del Código Fuente

Rational Team Concert permite administrar el código fuente en forma completa. En la figura 5.20. se muestra y explica gráficamente la forma en la que lo administra.

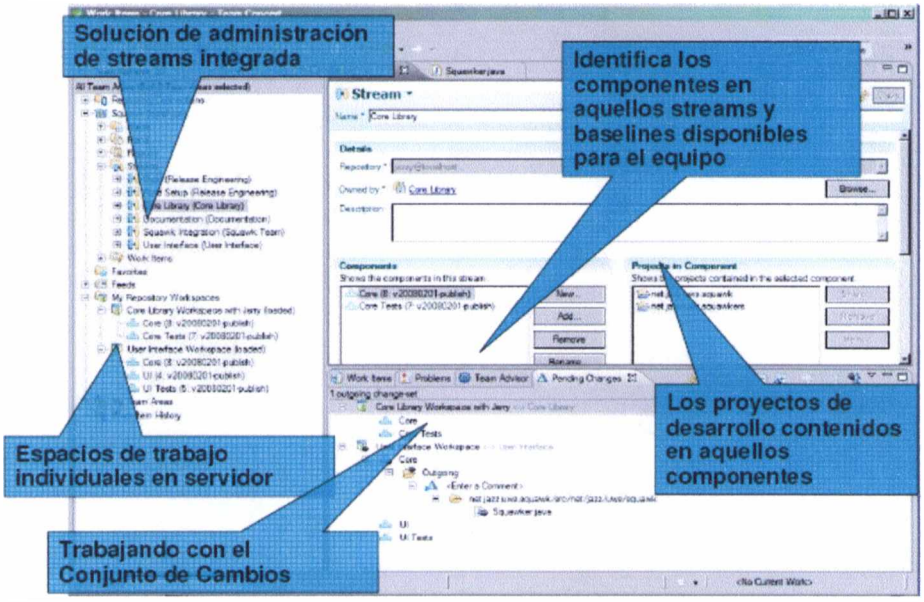

Figura 5.20. Administración de código fuente en RTC

5.2.6.6.5. Informes y Tableros – Visualizando el estado del proyecto

- Rational Team Concert utiliza el motor de informes BIRT (BIRT es un sistema de presentación de informes open source basado en Eclipse que se integra con una

aplicación Java/J2EE para producir reportes completos). En la figura 5.21. se explican gráficamente diferentes aspectos que considera RTC para Dashboards y Reporting.

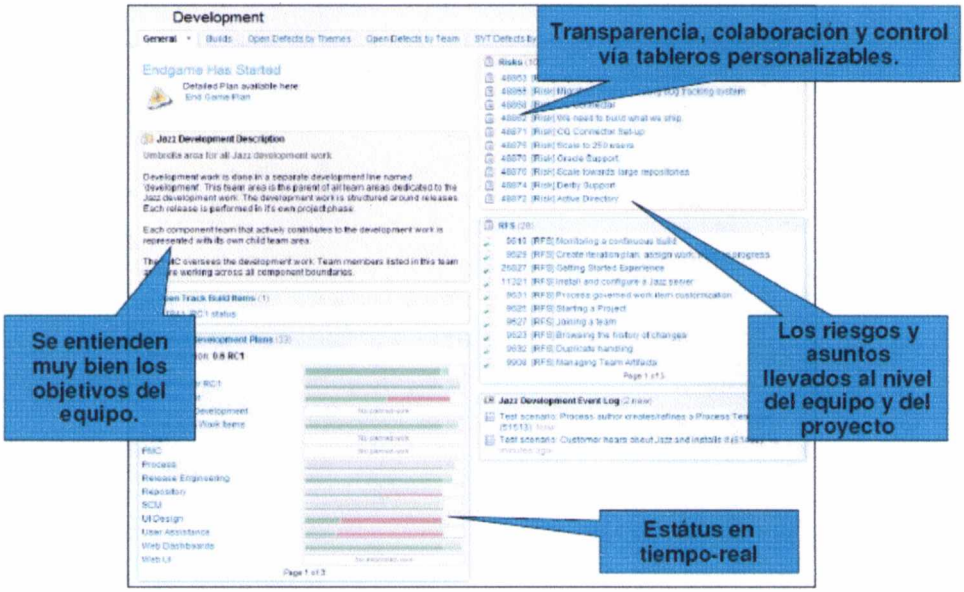

Figura 5.21. Dashboards y Reporting

- Una gran variedad de formatos de informe se han diseñado y están disponibles para mostrar una vista actualizada de los proyectos (Figura 5.22.):
  - Informes para el estado de los Builds
  - Informes para ver la carga del equipo y la distribución de los Work Items
  - Informes para el código
  - Etc.

Figura 5.22. Gran variedad de formatos de Informes

- Se pueden organizar datos como el detalle de los equipos, el estado de los objetivos actuales, etc. en Dashboards y Reporting (Figura 5.23.)

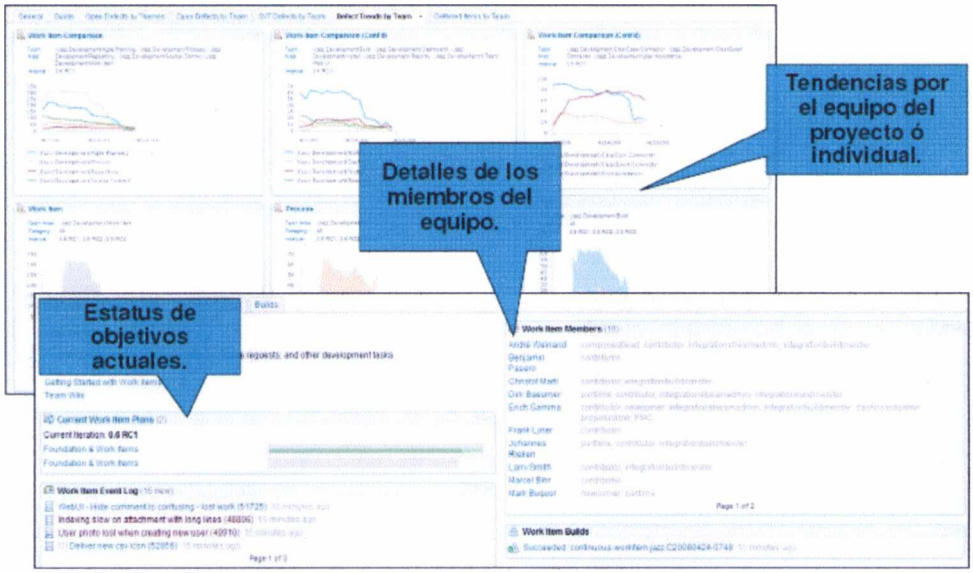

Figura 5.23. Organización de información en Dashboards y Reporting

- Los informes pueden organizarse en Dashboards en la UI Web (Figura 5.24)
- Los informes pueden ser exportados a los formatos Pdf, Xls, Doc, Ppt.

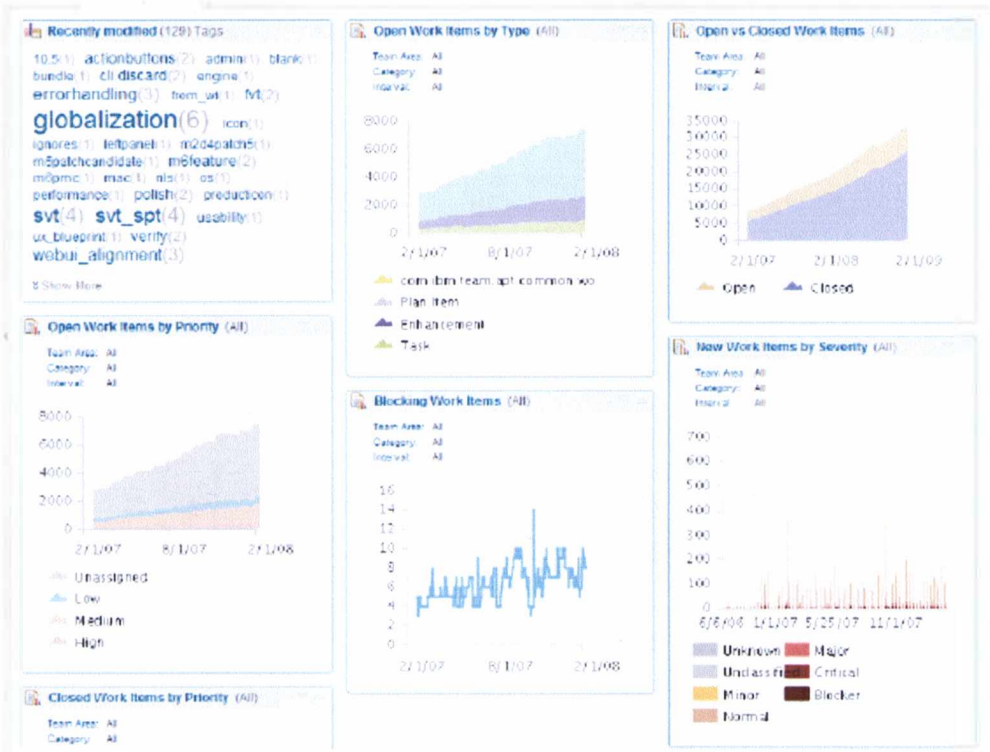

Figura 5.24. Dashboards en la UI Web

5.2.6.6.6. Anatomía Básica SCM en Jazz

Los Streams son para compartir recursos. Por ejemplo, un "Team Development Stream" contiene todos los artefactos que un equipo está desarrollando.

Un "Repository Workspace" es el espacio personal almacenado en el repositorio. Este almacena los cambios del desarrollador de forma intermedia y no es visible a otros miembros del equipo hasta que sea liberado al stream. El "Repository Workspace" contiene componentes. Jazz entiende la estructura del proyecto y creará proyectos Eclipse con estos componentes.
"Local Workspace" es donde se editan los recursos.

Los workspaces de Eclipse es donde se trabaja

Los cambios viajan en ambos sentidos.

En la Figura 5.25. se muestra gráficamente esta composición.

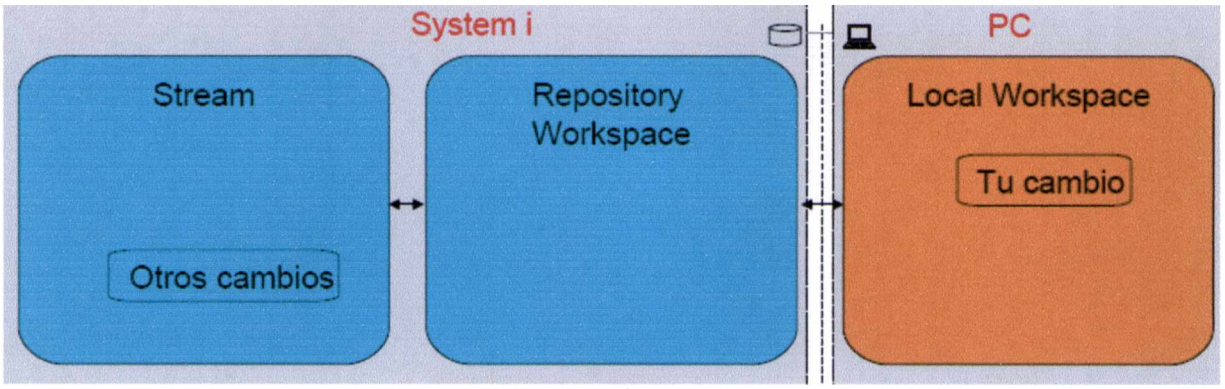

Figura 5.25. SCM Básico

Como complemento a este aporte, se enumeran brevemente los pasos y se presenta un ejemplo grafico (Figura 5.26.) sobre el flujo de un cambio desde el Local WorkSpace.

1. Crear Repository Workspace
2. Cargar dentro de Local Workspace
3. Un Desarrollador Edita un archivo
4. Se realiza un Check-in el nuevo archivo al Repository Workspace
5. Se realiza un Deliver al Steam para compartir con otros desarrolladores

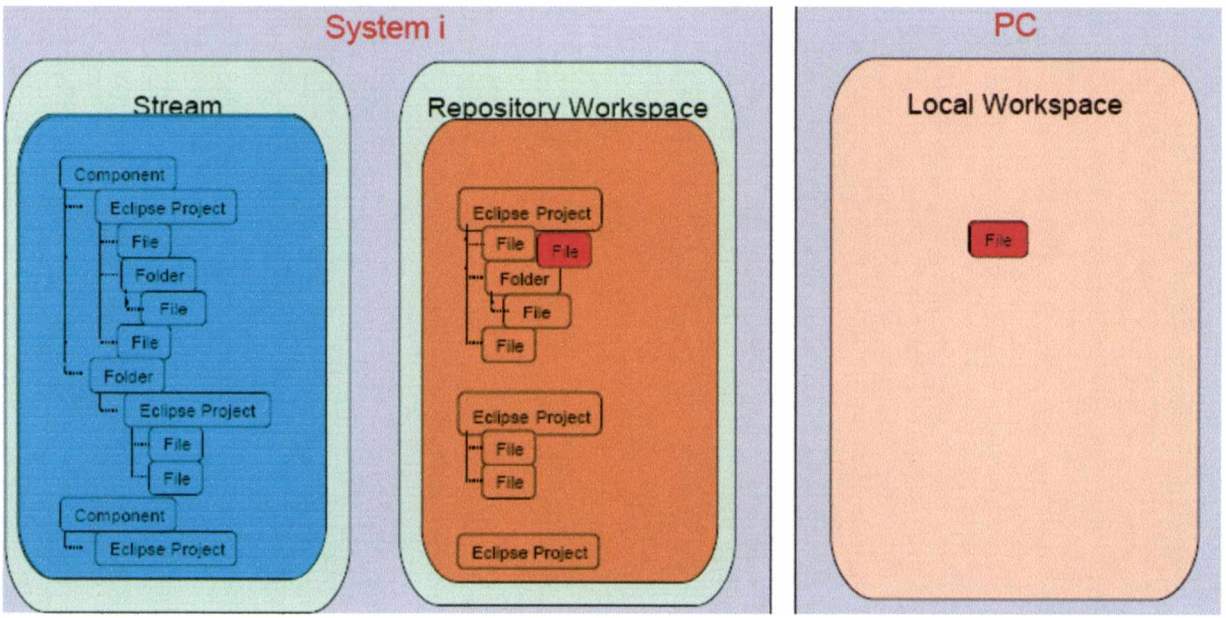

Figura 5.26. Flujo de un cambio desde el Local WorkSpace

5.2.6.6.7. Conjunto de cambios (Change-set)

- Compuesto de un conjunto de cambios en uno o más archivos y/o carpetas
  - Manejar las dependencias de forma integrada y transparente
  - Los Change-set son otro ejemplo de cómo Jazz comprende un proyecto

- Un Change-set que afecta a múltiples recursos se comportará como una unidad atómica única (hará commit solo si se pudieron realizar todos los cambios... realiza todos los cambios, o no realiza ninguno).

➢ Indicar la razón de los cambios
- Trazabilidad de los artefactos
- Colaboración a través de comentarios
- Hacer referencia al Work Item involucrado

➢ Puede ser compartido con otro miembro del equipo en tiempo real para aumentar la productividad
- Vía Stream
- Desde el repositorio del workspace, a través de un Work Item

En la figura 5.27. se muestra el camino típico que recorren los Change-Set desde el repositorio local hasta el "Stream Repository".

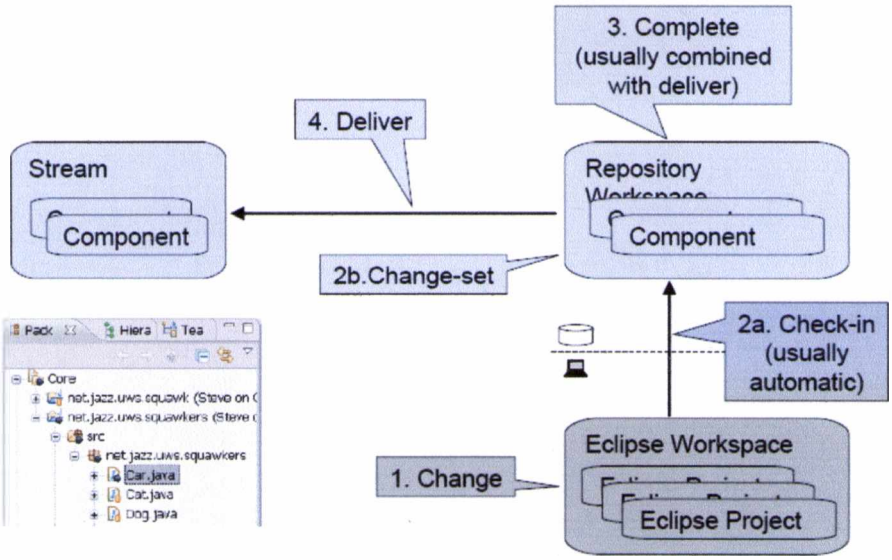

Figura 5.27. Change-Set desde el repositorio local

En la figura 5.28. se muestra el camino típico que recorren los Change-Set desde el "Stream Repository" hasta el repositorio local.

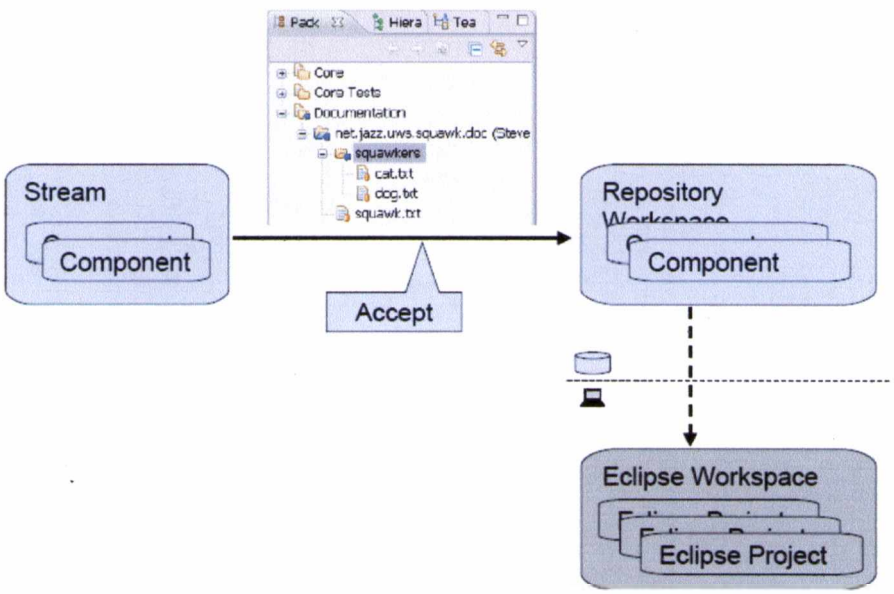

Figura 5.28. Change-Set desde el "Stream Repository"

5.2.6.7. Rational Team Concert Build (Compilación)

Los Build en el mundo del equipo de desarrollos ágiles (Figura 5.29.) tienen muchos desafíos a los que RTC intenta superar.

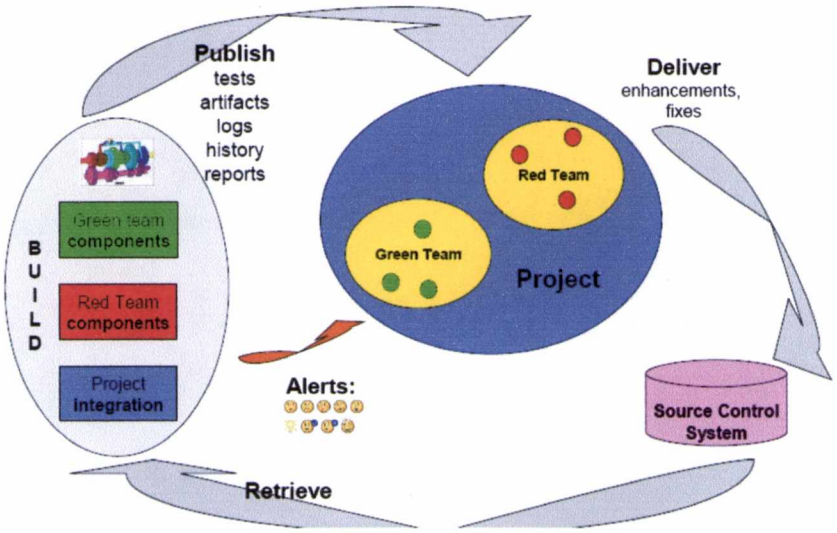

Figura 5.29. Builds en el mundo ágil

- *Rational Team Concert Build:*
  - Es una parte integral de la infraestructura del proyecto: Proceso consistente y repetible en todo el proyecto
  - Brinda conciencia a los desarrolladores sobre el progreso de construcción y los resultados: Fácil intercambio de información
  - Enlaces entre el Build de resultados y los artefactos relacionados de Jazz: Experiencia integrada, trazabilidad y seguimiento en tiempo real
  - Permite a los desarrolladores a tener un área de Build privado (Figura 5.30.): Construir y testear el código antes de entregar a la rama principal. Los Builds privados o personales permiten construir cambios antes de entregarlos al Stream. Esto da cierta seguridad de que los cambios no interrumpirán al equipo cuando se los aplique.

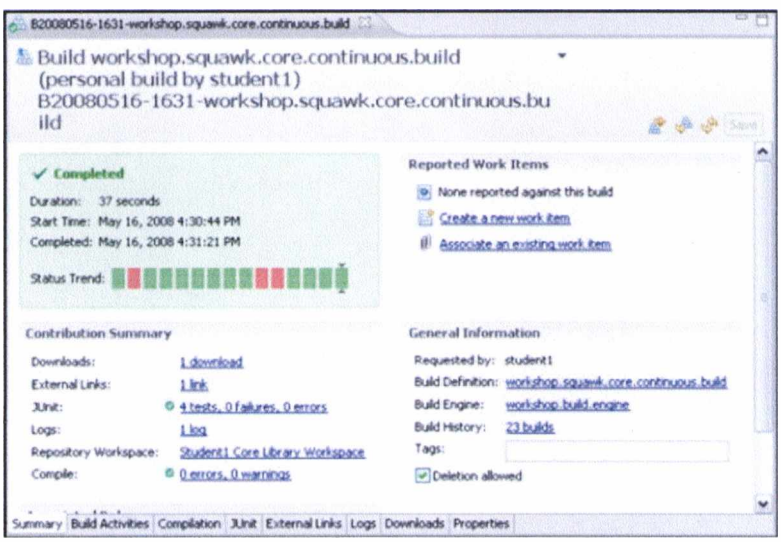

Figura 5.30. Builds Privados en RTC

- Adaptación a tecnologías Build existentes (Ant, CruiseControl, Build Forge, Maven): Aprovecha la tecnología que se adapte mejor a un proyecto

- Los Builds son muy visibles al usuario (Figura 5.31)

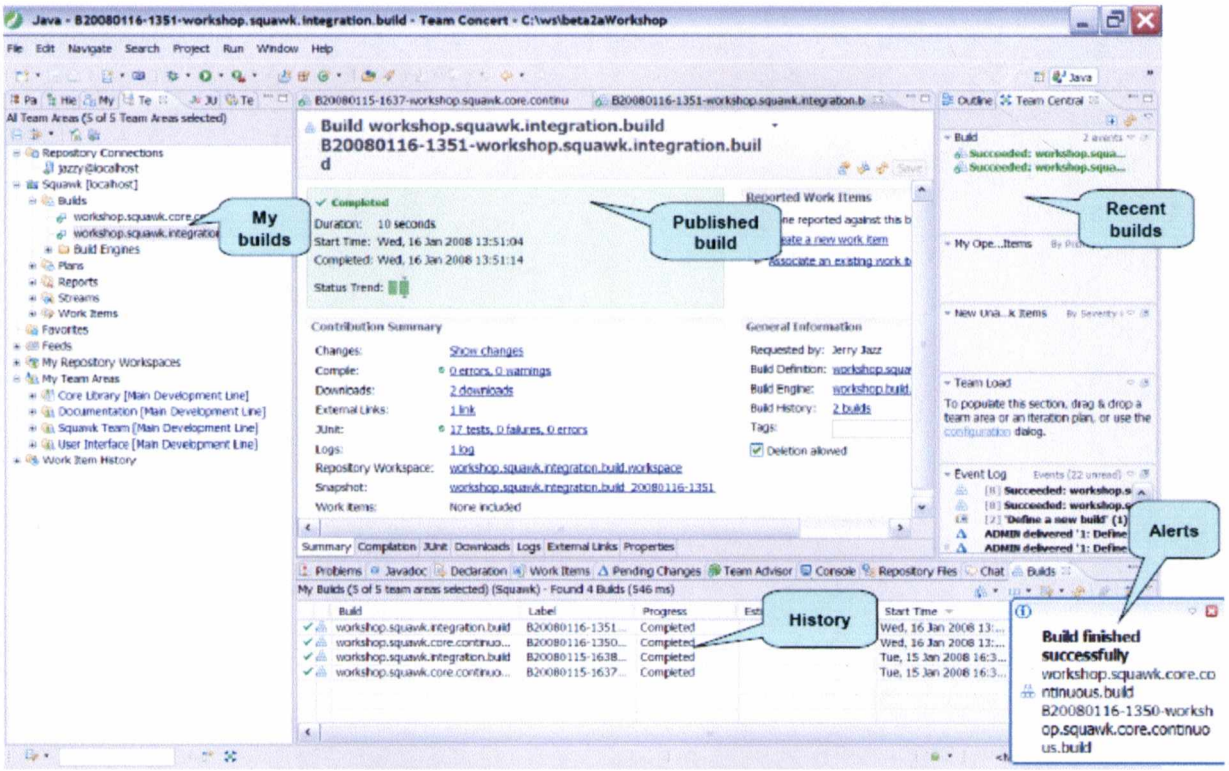

Figura 5.31. Builds visibles al usuario

- Un Build puede solicitar un Snapshot si hay problemas o algún cambio en un Build anterior (Figura 5.32).

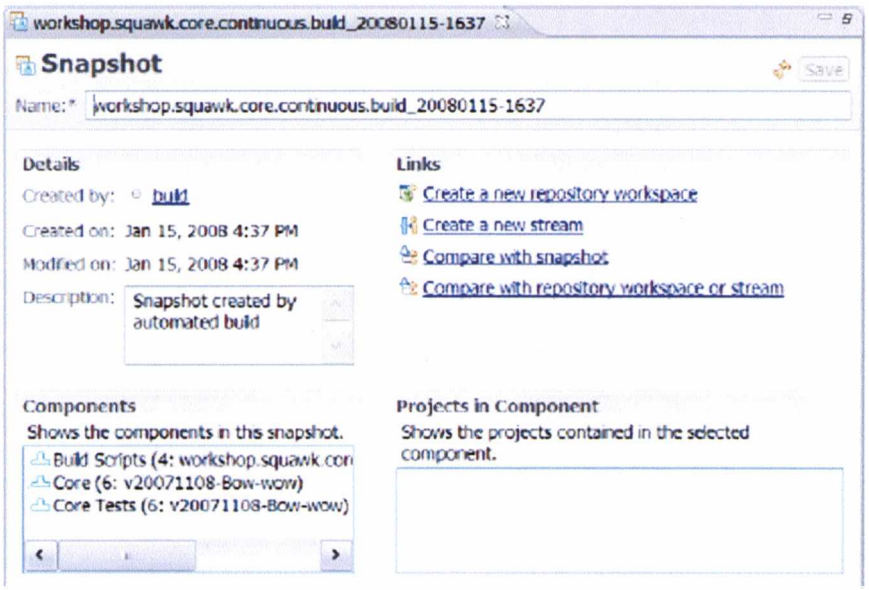

Figura 5.32. Snapshot solicitado por un Build

En la figura 5.33. se brinda una descripción general de Builds en RTC

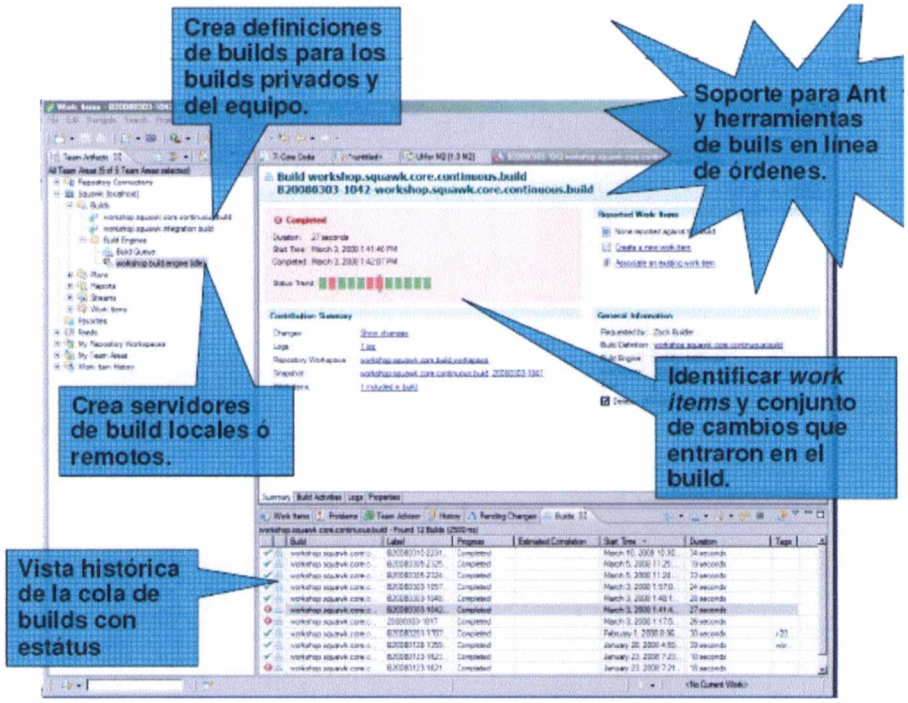

Figura 5.33. Builds en RTC

5.2.6.8. Cambios y Seguimiento

- **Build (Figura 5.34)**

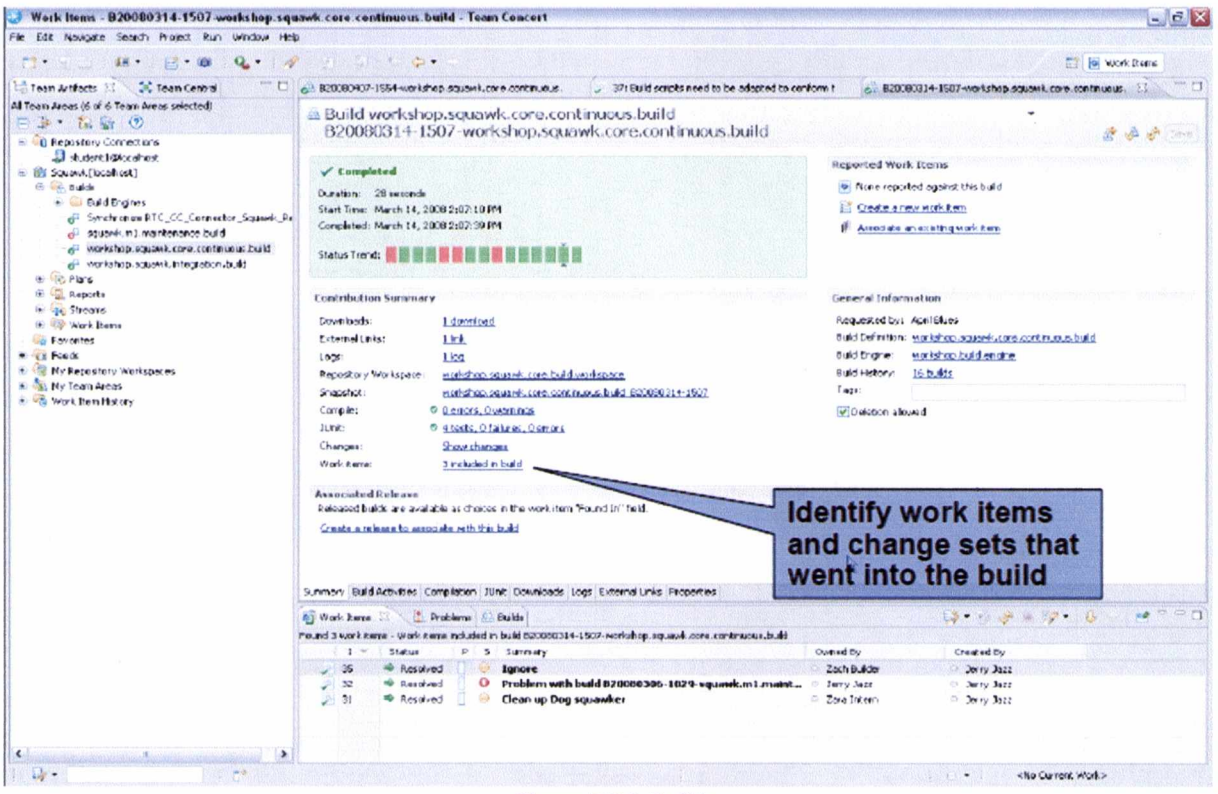

Figura 5.34. Builds

- **Work Items (Figura 5.35)**

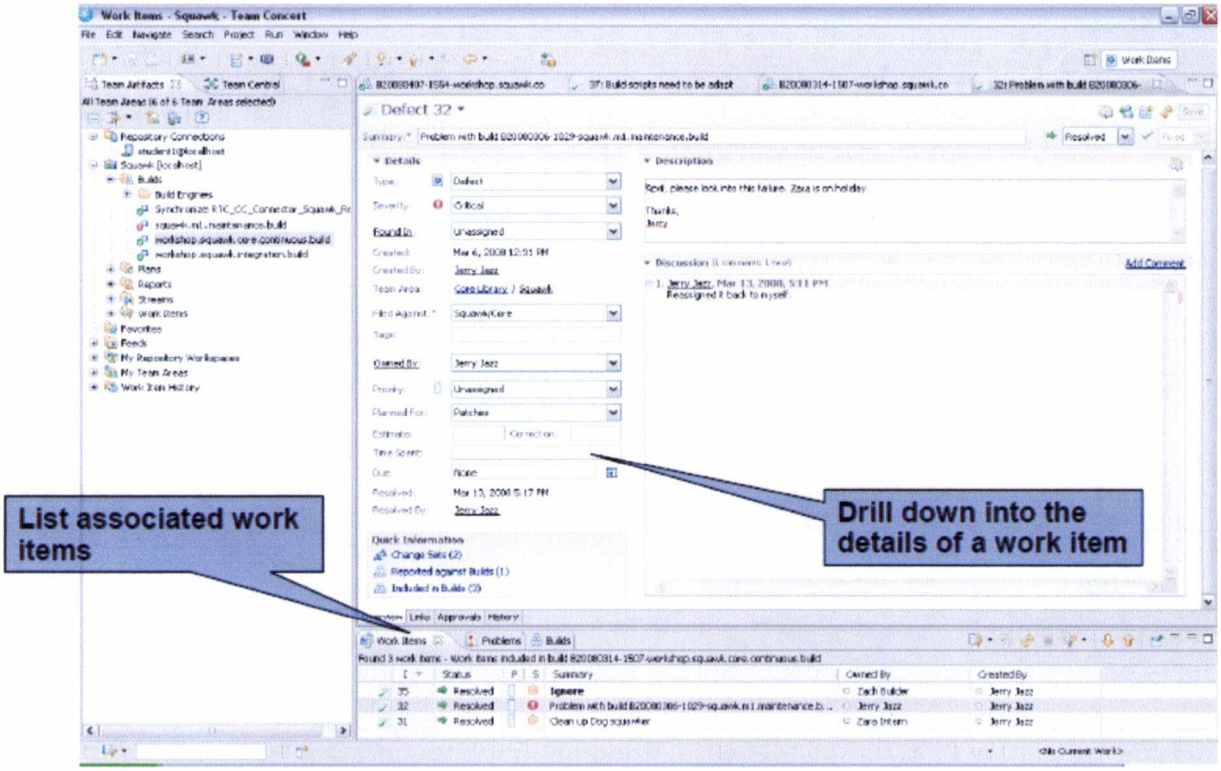

Figura 5.35. Work Items

- **Conjunto de cambios – Change-Set (Figura 5.36)**

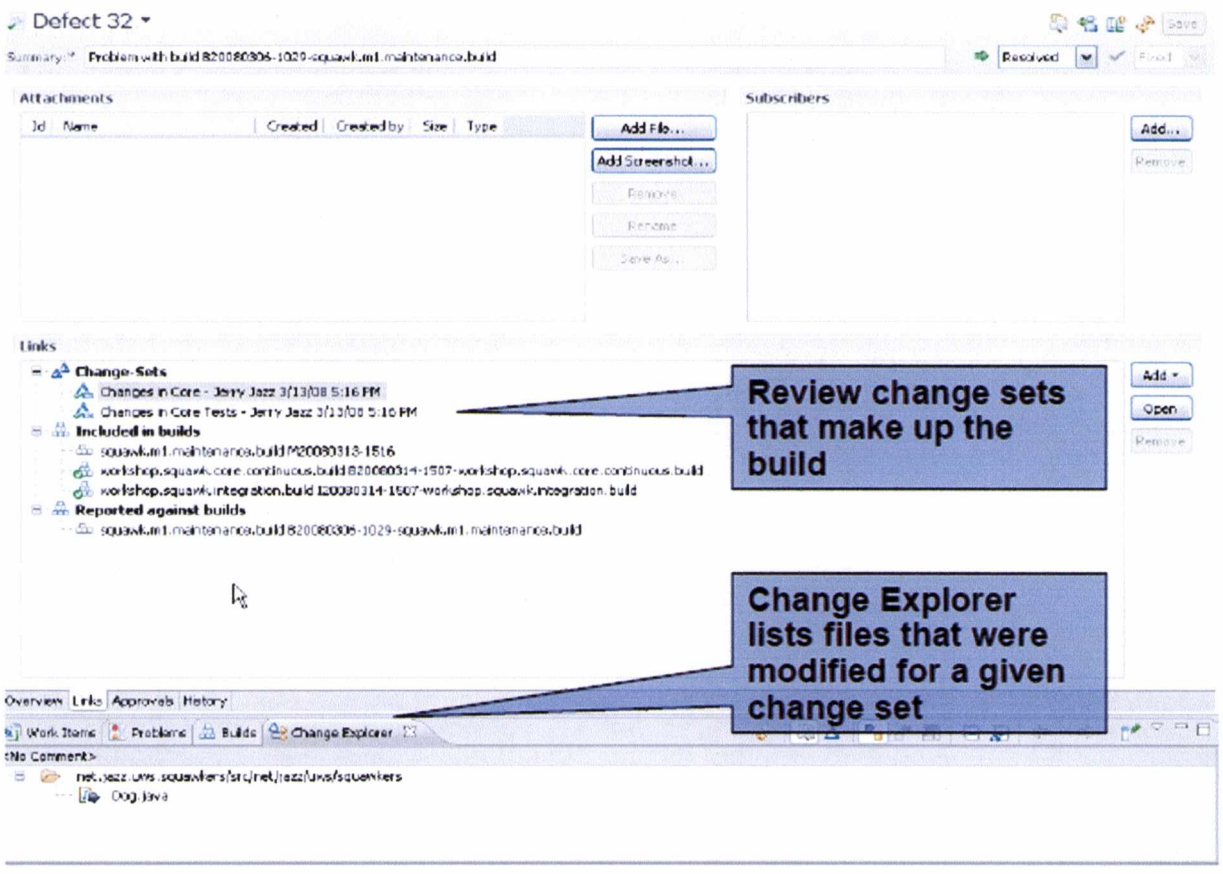

Figura 5.36. Conjunto de cambios

- **Comparación de cambios (Figura 5.37)**

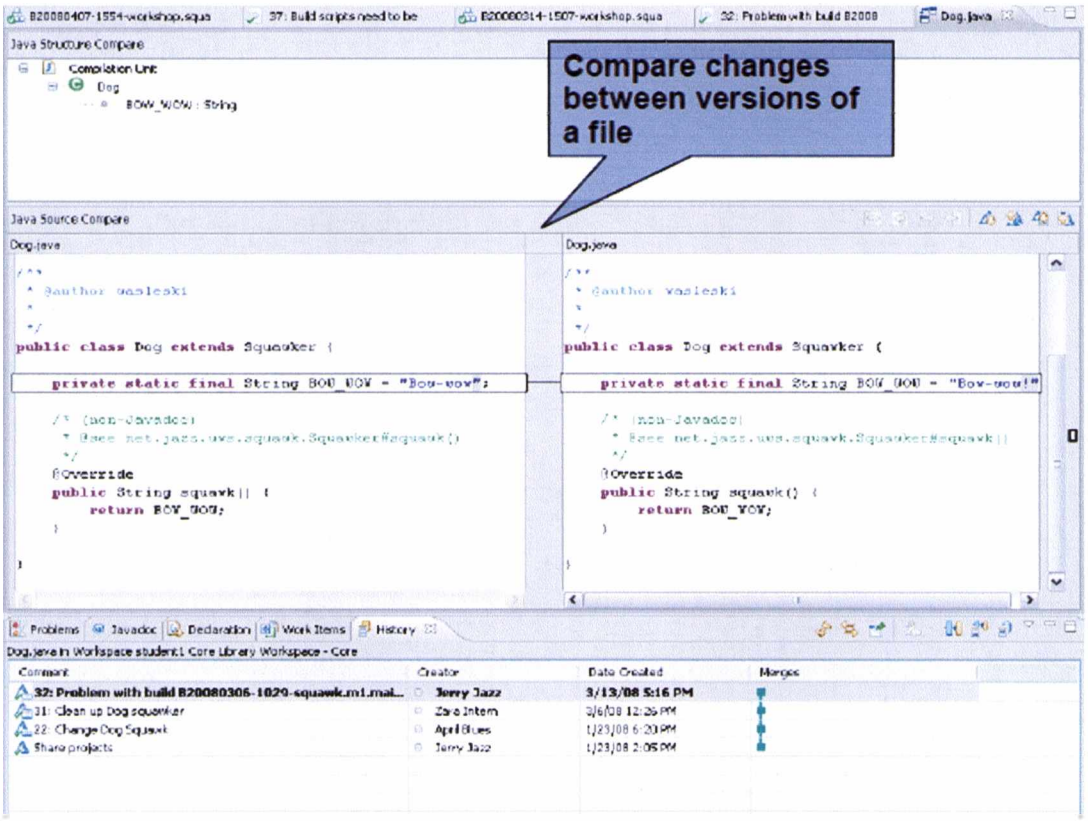

Figura 5.37. Comparación de cambios

- **Visualización de la historia de cambios (Figura 5.38)**

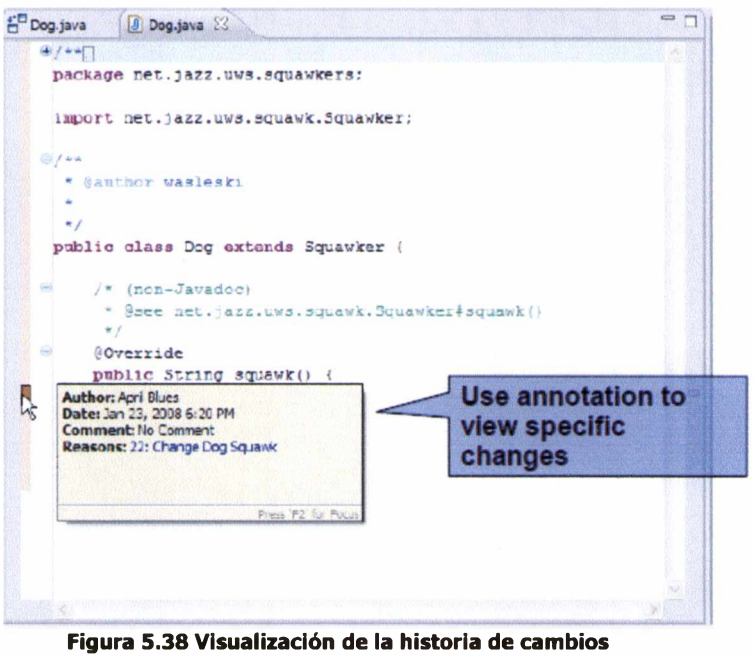

Figura 5.38 Visualización de la historia de cambios

5.2.7. Ejemplo de integración RTC, RRC, RQM

Como ya se explayó, Jazz es una plataforma que favorece la integración de diversas herramientas. Se proporcionará un ejemplo de integración entre las soluciones RTC, RQM y RRC (Figura 5.39.).

Repasando:

- **Rational Team Concert (RTC):** Solución para la gestión unificada del cambio, control de versiones y compilación que permite que diferentes equipos colaboren en tiempo real en el desarrollo de software
- **Rational Quality Manager (RQM):** Gestión unificada de todas las pruebas a través de una consola Web centralizada.
- **Rational Requirements Composer (RRC):** Solución para ayudar a los usuarios y analistas en la definición de los requisitos y mejorar la comunicación entre usuarios de negocio y el equipo de desarrollo

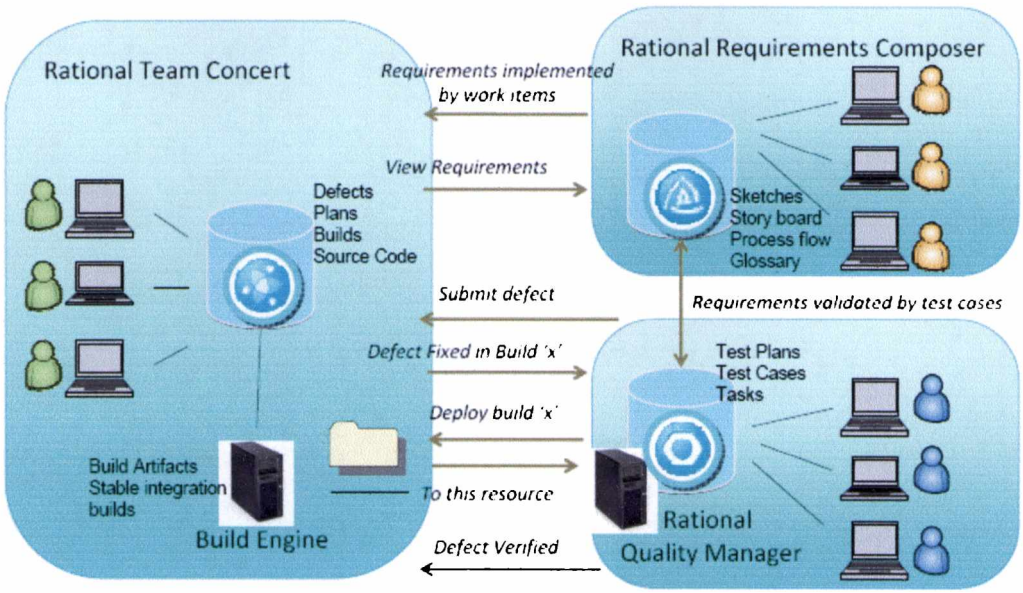

Figura 5.39. Integración RTC, RRC y RQM

- **Participantes del Escenario de Integración**
  - *Bob:* Analista de Negocio y Usuario de RRC
  - *Marco:* Líder de Desarrollo y Usuario de RTC
  - *Tanuj:* Líder de Pruebas y Usuario de RQM
- **Plan de Requerimientos**
  - Nombre sujeto a cambios
  - Un conjunto de requerimientos para una iteración o un release completo
  - Punto de integración: Enlace a un plan de iteración en RTC y un plan de pruebas en RQM
  - La integración puede ser alcanzada a través de este enlace
  - Ayuda a evaluar rápidamente el progreso global
- **Enlaces (RTC-RRC – Figura 5.40.)**
  - *Bob* enlaza a un plan de iteración en RTC
    - Relación uno a uno
    - Puede ser activado por *Marco* desde RTC
  - *Marco* enlaza un work item a un requerimiento
    - Navega hacia el requerimiento en RRC desde el plan

- Selecciona algún texto y crea un Work Item desde RTC (Alternativamente, enlaza a un existente work item)
- Los Requerimientos son implementados por work item(s)

- Característica del enlace
  - Es Bi-direccional (La eliminación también)

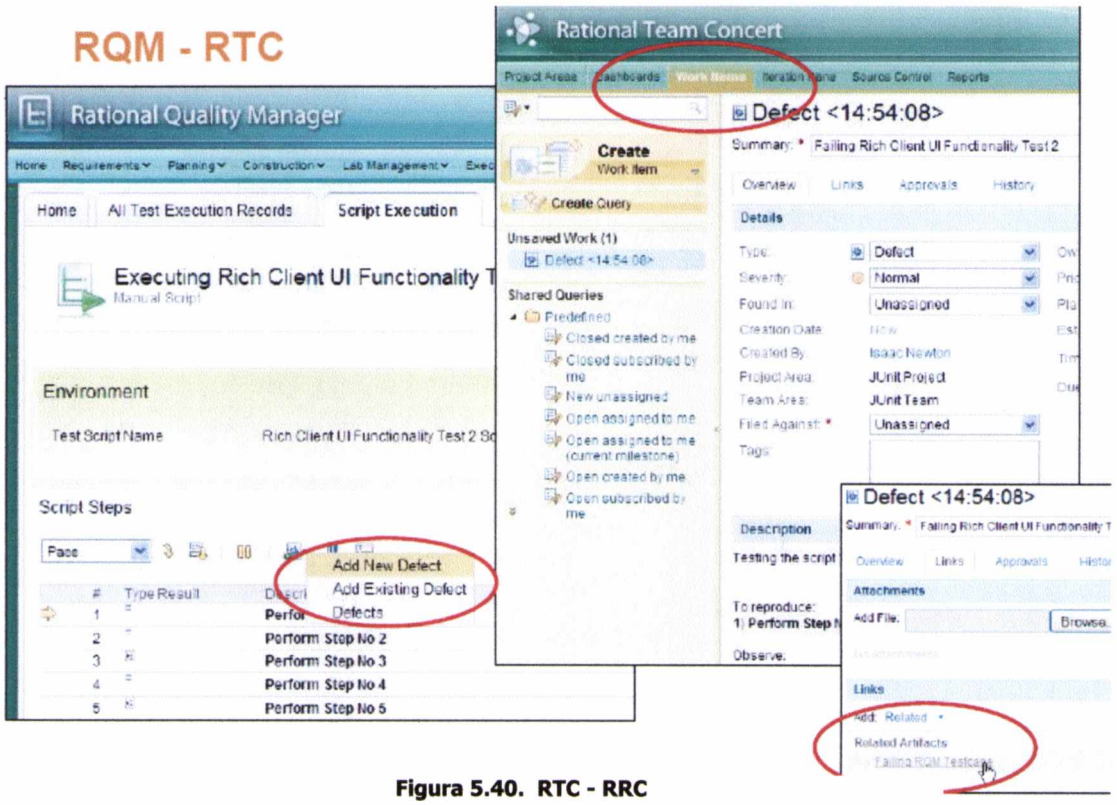

Figura 5.40. RTC - RRC

➢ **Comunicación Cross-product (RRC-RQM – Figura 5.41.)**

- Bob pregunta Tanuj si creó/actualizó algún caso de prueba
  - Actualiza el plan o un requerimiento en RRC
  - Crea una tarea para Tanuj en RRC
  - Alternativamente, Bob envía un email Tanuj desde RRC
  - Tanuj actualiza el caso de prueba y cierra la tarea
- Tanuj pregunta a Bob para clarificar el requerimiento
  - Navega hacia el requerimiento desde el Caso de Prueba o Plan de Pruebas
  - Crea un comentario a Bob sobre el requerimiento
  - Bob actualiza el requerimiento según observación y lo resuelve
  - Alternativa: Email

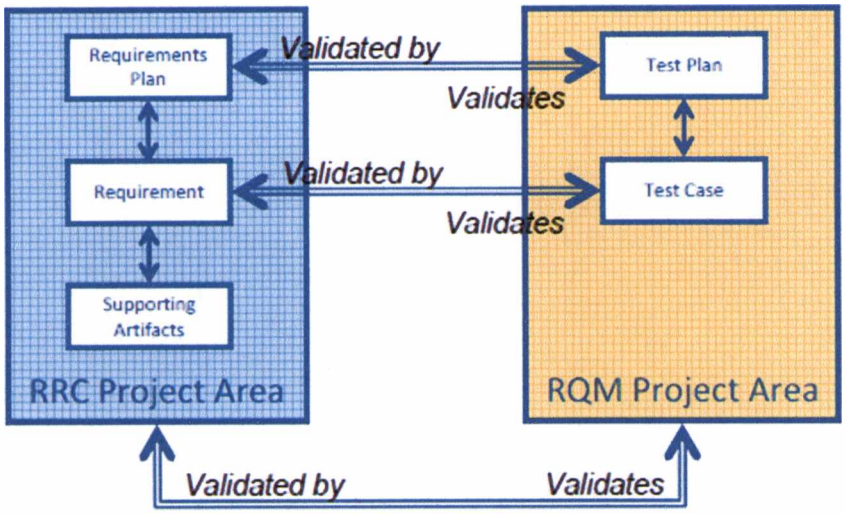

Figura 5.41. RRC - RQM

- **Métricas para medir el progreso**
  - **Uso de Dashboard**
    - Se pueden exponer artefactos de todos los productos
    - Personalizable
    - Queries predefinidos para identificar el progreso o identificar issues
      - Requerimientos con casos de prueba que fallaron
      - Requerimientos con Work Items no resueltos
      - Requerimientos no probados
  - **Plan de Requerimientos**
    - Página de Status
    - Queries predefinidos
    - Progreso completo o por requerimiento: Implementación: 60% (6 of 10)

5.2.8. Ventajas de RTC

En la figura 5.42 se muestran algunas de las ventajas más importantes de RTC.

Fácil de usar	Plantillas de procesos predefinidas, facilitan el uso
Fácil de integrar	Combina tres funciones para la gestión del ciclo de vida de aplicaciones en una solución integrada
Fácil de manejar	Integra equipos de hasta 50 miembros (o más) en un único servidor basado en un middleware común.
Gestión de Work Items integrada	Crea y sigue automáticamente el progreso de los work items individuales en concordancia con el proceso de equipo y con las reglas del proyecto. Además captura y mantiene las relaciones entre los work Items y otros artefactos como los change-sets y los builds
Control de código fuente integrado	Proporciona un sistema de control de

	versiones esencial, gestión del workspace, y soporte de desarrollo paralelo para personas y equipos. Además es plenamente integrado con los Work Items (por que se hizo un cambio en el código?) y al gestor de compilación (cuando se compilo el código?)
Gestión de compilación integrada	Ordena y ejecuta procesos de compilación de software. Aprovecha múltiples servidores para el procesamiento de los Build. Integrado con los Work Items y el control de código fuente para proporcionar información detallada de los artefactos involucrados para garantizar el éxito en las compilaciones.
Integración y soporte de ClearCase & Subversion	Se integra manteniendo los Work Items y la gestión de Builds
Integración y soporte de ClearQuest	Se integra manteniendo los Work Items y la gestión de Builds, y el control de código fuente.
Process awareness y Automatización	Provee variedad de plantillas de procesos personalizables, incluyendo SCRUM, que guían el flujo de trabajo y automatiza los pasos del proceso y checkpoint con un grado de rigor variable.
Team awareness y Colaboración	Proporciona presencia en proyectos integrados y mensajería
Soporte a Visual Studio	Proporciona acceso al sistema de control de versiones de Team Concert, a los work item, y a los Buids, para desarrolladores nativos de Visual Studio, fomentando la unificación de los equipos de desarrollo.
Colaboración de documentos	Integra colaboración de software y documentos a través de la integración con Microsoft SharePoint o Lotus Quickr

Figura 5.42. Ventajas de RTC

Capítulo 6

Análisis de la herramienta RTC mediante un caso concreto basado en Scrum y OpenUp

En este capitulo se explaya sobre el desarrollo de un caso de estudio concreto utilizando la herramienta RTC. Se propone el desarrollo de un software de pequeña/mediana envergadura para analizar las características generales de la herramienta RTC (también características puntuales).

Se decidió desarrollar el caso de estudio utilizando y analizando exhaustivamente el proceso Scrum, ya que éste, es uno de los procesos ágiles más utilizados en la actualidad. Finalizado esto, se realizaron pruebas en forma más genérica del proceso OpenUp. Cabe aclarar que la mayoría de las pruebas acerca de las funcionalidades de RTC, están en el apartado dedicado a Scrum (6.5. Caso de estudio basado en Scrum). No se creyó necesario iniciar un proyecto completo en OpenUp, ya que, si bien existen diferencias en la definición de los templates, y otras diferencias menores, no existen diferencias en las funcionalidades que brinda RTC.

En este apartado se compartirá la experiencia de utilizar RTC, se exponen características del software, problemas, novedades, ventajas etc.

6.1. Sobre el caso de estudio

Para probar la herramienta RTC, se optó por el desarrollo de una aplicación WEB.

Se trata de una aplicación para la gestión y control de ventas, alquileres, propiedades y lotes.

Las operaciones que deben almacenarse son las ventas y alquileres.

Las ventas se realizan de distintas formas: contado, financiadas. Para esto último se construyen planes de pagos de los cuales también se lleva un registro de ellos. Estos planes de pagos se pueden cancelar, rescindir y refinanciar. Dichas operaciones deben registrarse.

El sistema para administrar las ventas y los planes de pago mencionados necesitan almacenar datos de las propiedades, personas involucradas, escribanos que intervienen en escrituras.

En los alquileres deben registrarse los pagos mensuales durante el tiempo del alquiler.

La aplicación además debe contar con la posibilidad de imprimir balances anuales y mensuales; para ello también es necesario almacenar gastos.

El proyecto se desarrolla bajo una aplicación Web escrita en Django (Python Web Framework), utilizando PyDev para la integración con eclipse.

6.2. Conexiones de los usuarios

Luego de instalar tanto el servidor como el cliente, y de realizar pruebas exploratorias del software, se comenzó con el desarrollo de la aplicación WEB elegida.

Cuando se instala el servidor, existe un único usuario por default, que es "ADMIN". Se ingresó entonces con ese usuario, y se crearon los usuarios iniciales para utilizar la aplicación (Inicialmente los nombres de usuario coinciden con las contraseñas... luego cada usuario puede modificar su contraseña). Esta tarea, se puede realizar, tanto desde la interfaz WEB como desde el cliente, en nuestro caso, RTC integrado en Eclipse.

La mayoría de las pruebas que se mostrarán fueron realizadas en el cliente, pero es necesario aclarar en este punto que casi la totalidad de las funcionalidades que provee RTC pueden aprovecharse también en el servidor.

El primer paso para utilizar el software RTC Cliente consiste en la configuración de las conexiones de los usuarios. El usuario correspondiente, desde su terminal, debe ingresar sus datos y los del servidor, indicando la URL donde esta alojado el servidor, su nombre y su contraseña. Esta primera acción se realizó de forma espontánea y natural; no fue necesario recurrir a ninguna ayuda. Se probaron conexiones LAN, y a Internet, incluso ubicando al servidor en una terminal con acceso Wi-Fi. En todos los casos, el tiempo de respuesta resultó ser muy satisfactorio. También se configuro el servidor para que consuma no más de 200 Megas de RAM, y continuó con funcionamiento prácticamente óptimo. En la figura 6.1. se muestra este paso.

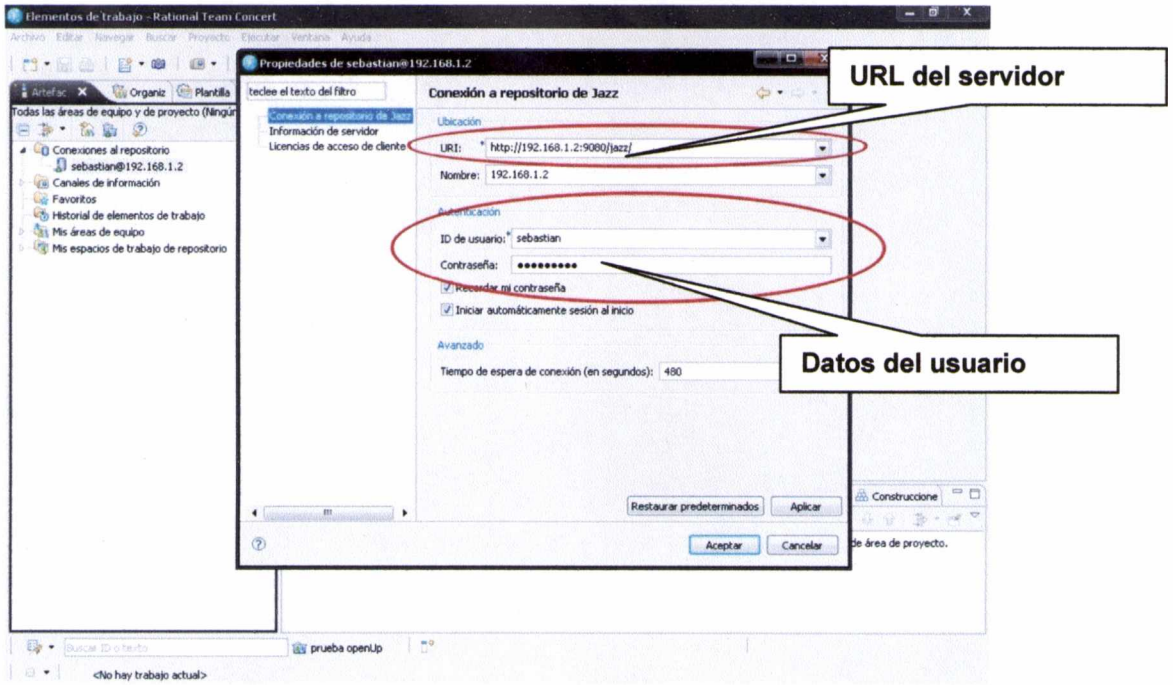

Figura 6.1. Conexión de Usuario

6.3. Creación de usuarios

El segundo paso que se realizó, consiste en la creación de usuarios que utilizaron el software.

En esta parte un administrador crea usuarios indicando un ID y un mail respectivo.

Se debe determinar además el tipo de usuario del cual se trata. Los tipos de usuarios posibles son los siguientes:

- **JazzAdmins**: Administrador de un repositorio Jazz con total acceso de lectura y escritura.
- **JazzDWAdmins**: Administrador de un repositorio Jazz con permisos específicos para controlar el *Data Warehouse* del servidor Jazz.
- **JazzGuests**: Usuarios con acceso de *solo lectura* al repositorio Jazz.
- **JazzUsers**: Usuarios con acceso de lectura/escritura regular para acceder al repositorio Jazz.

También se debe determinar el tipo de licencia de acceso de cliente que se asignará al usuario. En la figura 6.2. se detallan las posibles licencias que se pueden otorgar:

No Client Access License	• Read access to all capabilities
Contributor	• Contributor has read-only access to all capabilities • Contributor can create and modify work items unless otherwise restricted by process permissions • Contributor can create and save work item queries • Contributor can view team dashboards and other users' dashboards, but cannot create a personal dashboard • Contributor can view reports created by other users, but cannot create or modify reports • Contributor Client Access Licenses do not add to the server user limit. You can add Contributor Client Access Licenses even if the server user limit has been reached.
Developer	• Developer has read access to all capabilities • Developer has write access to all capabilities, unless otherwise restricted by process permissions • The total number of all users with Developer Client Access License cannot exceed the server user limit.
Build System	• The Build System Client Access License is for assignment only to a user ID used by a Build System. This enables the devices in the Build System to have read access to all capabilities, and write access to all capabilities unless otherwise restricted by process permissions. • Build System user IDs do not add to the server user limit.
ClearCase Connector	• Standard and Enterprise editions only • The ClearCase Connector Client Access License is for assignment only to the synchronization process account (the account used by a build engine that creates and synchronizes a ClearCase

	Synchronized Stream). • These user IDs do not add to the server user limit.
ClearQuest Connector	• Standard and Enterprise editions only • The ClearQuest Connector Client Access License is for assignment only to the single user ID being used by the ClearQuest Connector. The capabilities are restricted to only those capabilities needed by the ClearQuest Connector. • These user IDs do not add to the server user limit.

Figura 6.2. Licencias de acceso cliente

Hay otras posibles acciones que se pueden realizar en este momento pero que en nuestro caso lo configuramos mas adelante:

- Asociar a un área de equipo
- Determinar el entorno del trabajo
- Configurar ausencias planificadas
- Configuración de correo

Se puede configurar la aplicación para que los usuarios reciban la notificación por mail para alertarlos que fueron dados de alta en el sistema.

En una primera instancia se crearon 5 usuarios para el caso de Scrum; luego se añadieron más usuarios para el caso de OpenUp.

En la figura 6.3 se muestra gráficamente la creación de un usuario.

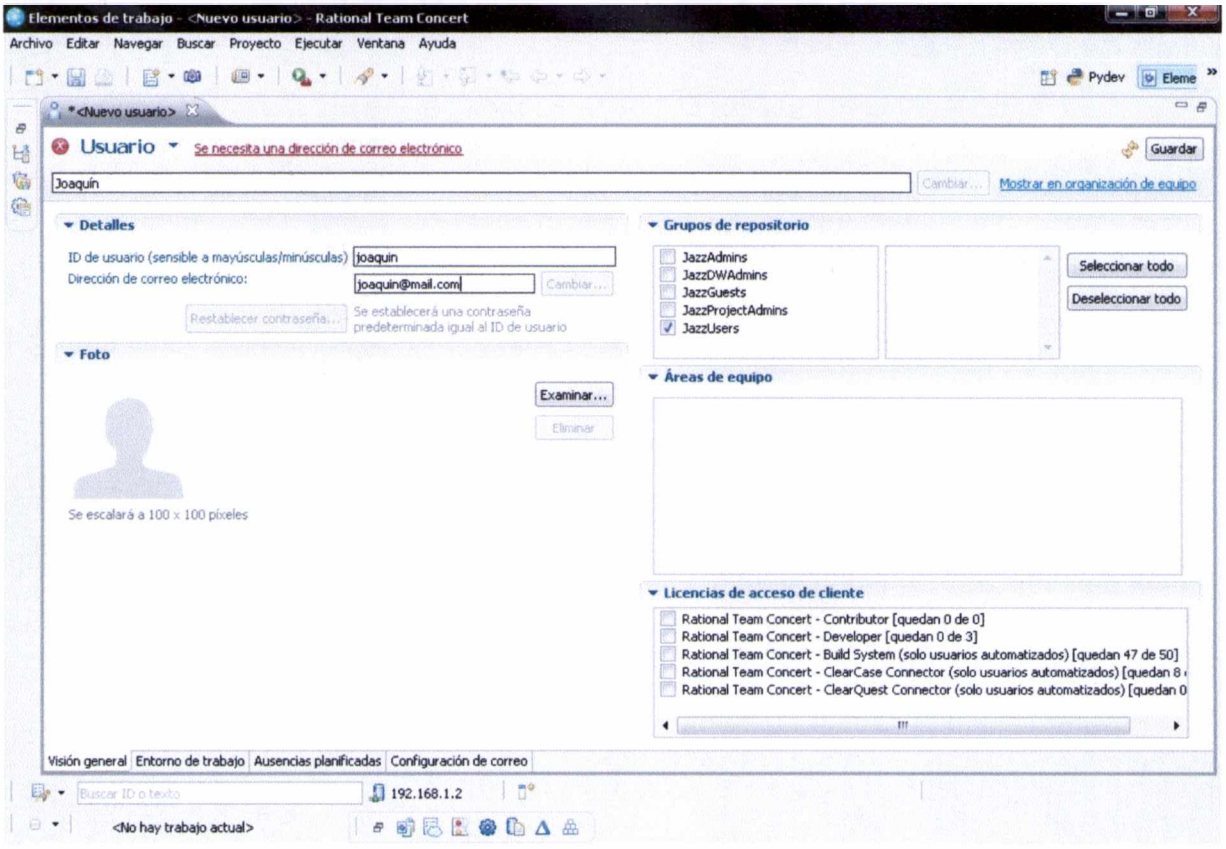

Figura 6.3. Creación de usuario

6.4. Creación de área de proyecto

Luego de crear los usuarios, se procedió a la creación del área de proyecto. Es en éste paso donde se opta por el template del proceso que se va a utilizar.

El área de proyecto es la representación por parte del sistema de un proyecto de software. El área de proyecto define las entregas de producto, estructura del equipo, proceso y agenda o programa del proyecto.

Un área de proyecto se almacena como de máximo nivel o elemento de root en un repositorio. Un área de proyecto hace referencia a los artefactos del proyecto y almacena las relaciones entre dichos artefactos. El acceso a un área de proyecto y a sus artefactos está controlado mediante permisos. Un área de proyecto no se puede borrar del repositorio; sin embargo, se puede archivar, lo que la situará en un estado inactivo.

Este paso es muy simple, pero se quiso plasmar ya que es el punto de inflexión entre el caso de estudio que se realizo utilizando Scrum y el que se realizo utilizando OpenUp. Hay plantillas de otros procesos, e incluso se pueden crear plantillas personalizadas.

En una primera instancia solo se debe definir un nombre al área de proyecto y un resumen, como lo muestra la figura 6.4.

Hecho esto, se debe seleccionar la plantilla que desea utilizarse... puede ser una plantilla predefinida o una plantilla personalizada. También existe la posibilidad de personalizar una plantilla predefinida, revisando la especificación de los procesos acorde al grupo de personas que utilizarán el software.

En nuestro caso primero se seleccionó la plantilla Scrum y luego la plantilla OpenUp. En la figura 6.5 puede observarse este paso.

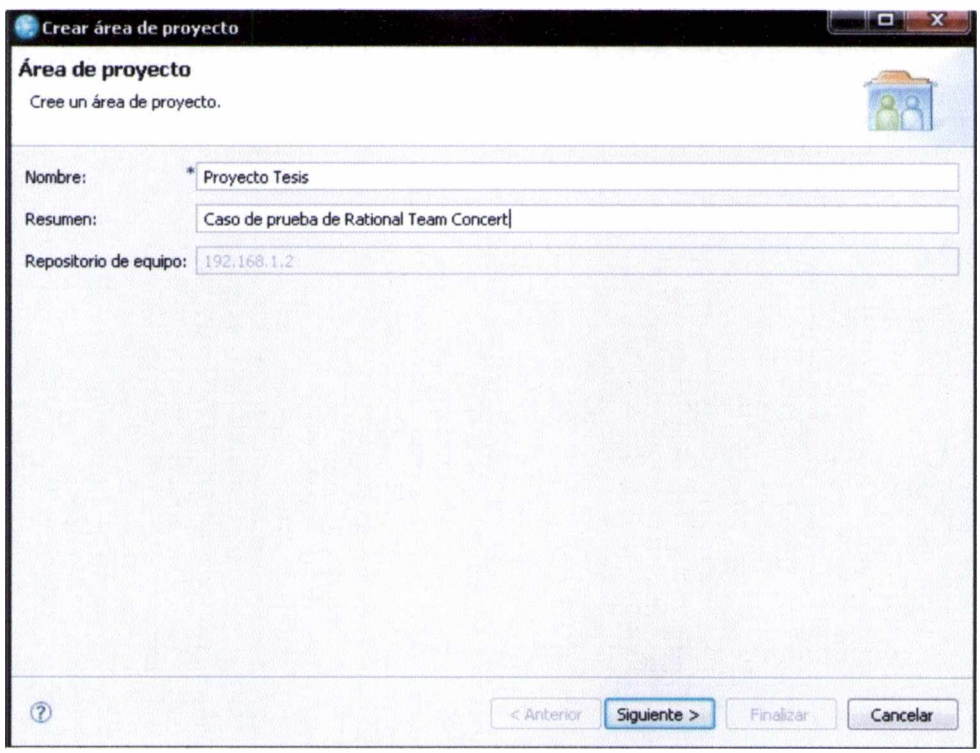

Figura 6.4. Definir área de proyecto

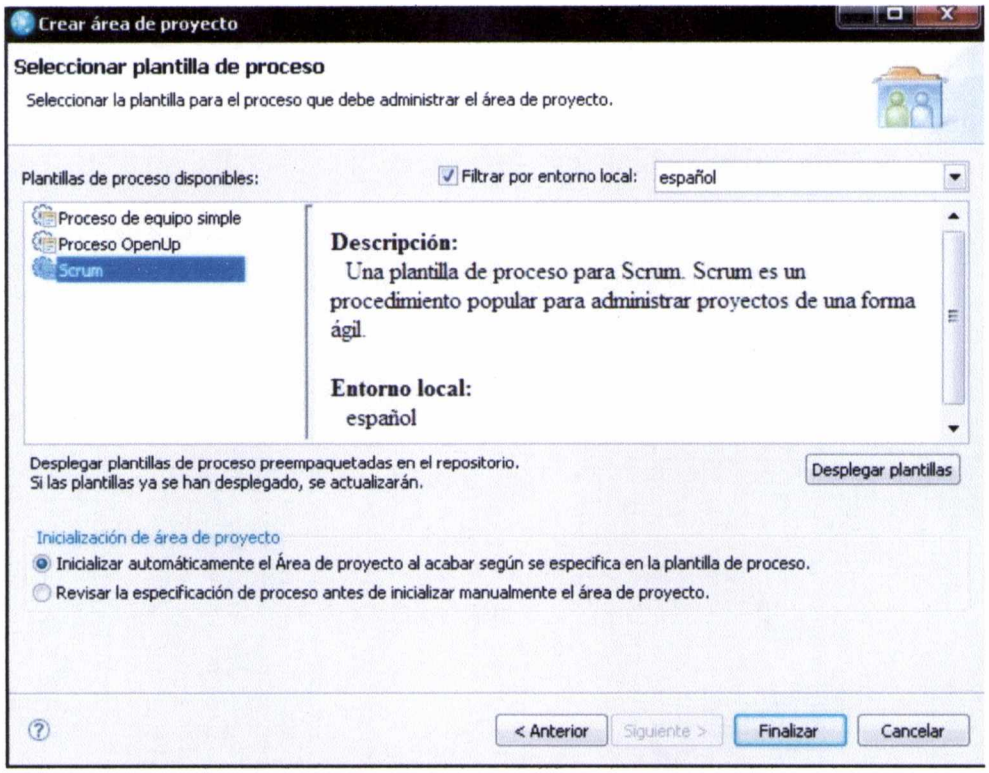

Figura 6.5. Definir plantilla de proceso

6.5. Caso de estudio basado en Scrum

En este apartado se explicarán las pruebas realizadas específicamente utilizando el template del proceso Scrum.

Como primer reacción, el simple hecho de utilizar Scrum desde posiciones remotas es contraria a su propia definición. Por definición, Scrum sugiere que "Todos los miembros del equipo trabajan en la misma localización física, para poder maximizar la comunicación entre ellos mediante conversaciones cara a cara, diagramas en pizarras blancas, etc. De esta manera se minimizan otros canales de comunicación menos eficientes, que hacen que las tareas se transformen en un "pasa pelota" o que hacen perder el tiempo en el establecimiento de la comunicación (como cuando se llama repetidas veces por teléfono cuando la persona no está en su puesto)."

Es importante remarcar que varias de las pruebas que aquí se realizaron, para mostrar las funcionalidades del software RTC, están disponibles también para el template del proceso OpenUp, ya que estas características son indistintas a la plantilla utilizada.

6.5.1. Configuración del área de proyecto

En este paso se procedió a configurar en área de proyecto.

Esta tarea se divide en 3 partes fundamentales. La configuración general, la configuración de las iteraciones y la configuración de proceso.

6.5.1.1. Configuración general

En la configuración general, habiéndose definido en un paso previo el nombre y el resumen del área de proyecto, se definió una descripción, los miembros y los administradores.

Como una característica importante se debe remarcar que desde esta sección se puede acceder a la plantilla de Scrum, revisar su especificación, y modificarla según las necesidades del equipo.

RTC también permite crear nuevos tipos de elementos de trabajo o modificar los existentes. Permite crear estados para cada Works Items y definir las transiciones válidas para cada elemento.

En nuestro ejemplo se optó por agregar un único miembro al área de proyecto con el rol de ScrumMaster (Usuario: Lucrecia), el cual también es administrador. En la figura 6.6. puede verse este paso.

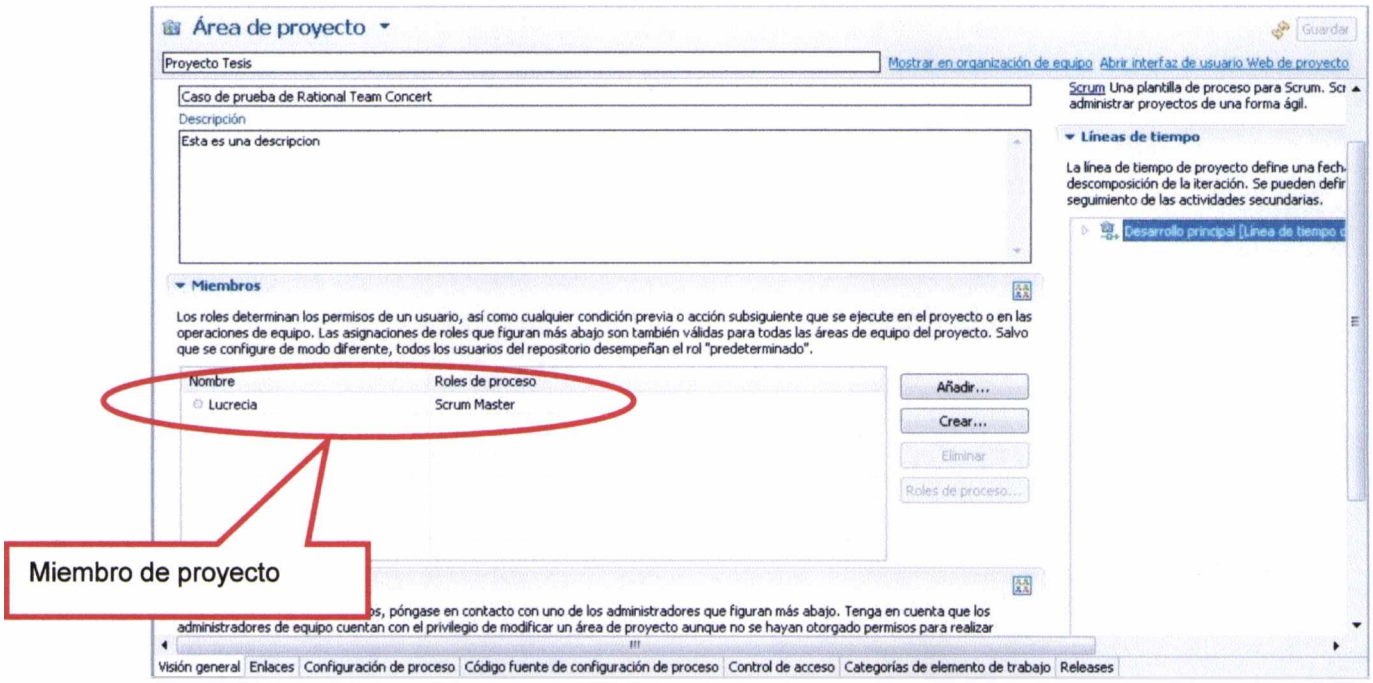

Figura 6.6. Configuración general de área de proyecto

6.5.1.2. Configuración de sprint

La configuración de las iteraciones consiste en definir, en primera instancia, la cantidad de sprint (iteraciones en Scrum) que tendrá el desarrollo, un nombre para cada sprint, un identificador, y la fecha de inicio y final de cada iteración.

En nuestro caso se propusieron 2 Sprint de entre 5 y 8 días como lo muestra la figura 6.7.

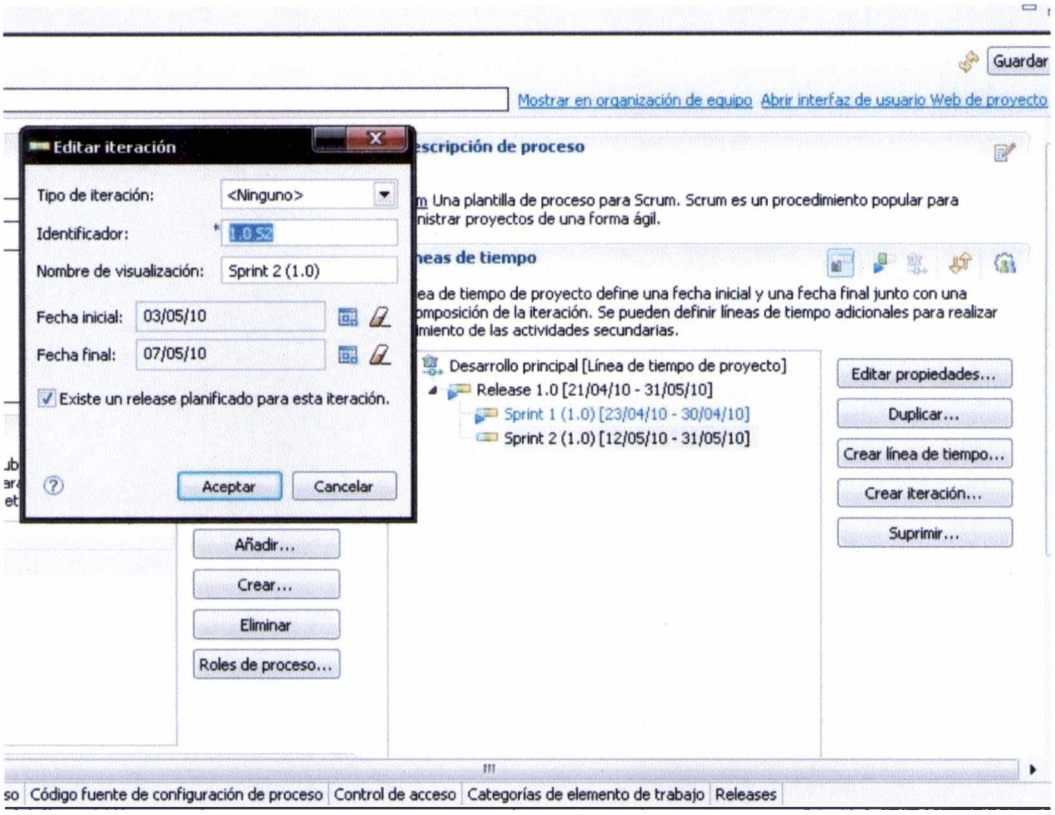

Figura 6.7. Configuración de Sprint

6.5.1.3. Configuración de permisos

Luego se configuró el proceso. Hay muchas y diversas opciones de configuración: Configuración respecto a los roles, configuración respecto al equipo, configuración respecto al proyecto, configuraciones de permisos, configuraciones de los sprint, hasta configuraciones de los miembros por cada sprint en particular, etc. Es importante remarcar que, por la experiencia que se adquirió al realizar esta prueba, se verificó que estas configuraciones constituyen una importante ventaja del software RTC. En la figura 6.8 se muestran algunas de estas opciones.

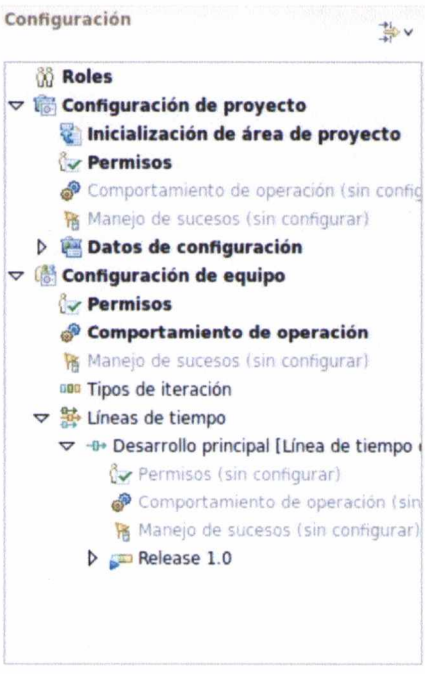

Figura 6.8. Configuración de Proyecto

Se configuraron los permisos que tienen los usuarios según su rol dentro del área de proyecto. Existen diferentes vistas lo cual facilita al usuario setear los permisos con la vista mas apropiada según su gusto. Se pueden mostrar "acciones por rol" o "todos los roles y todas las acciones" en una tabla. En la figura 6.9 se muestra este paso.

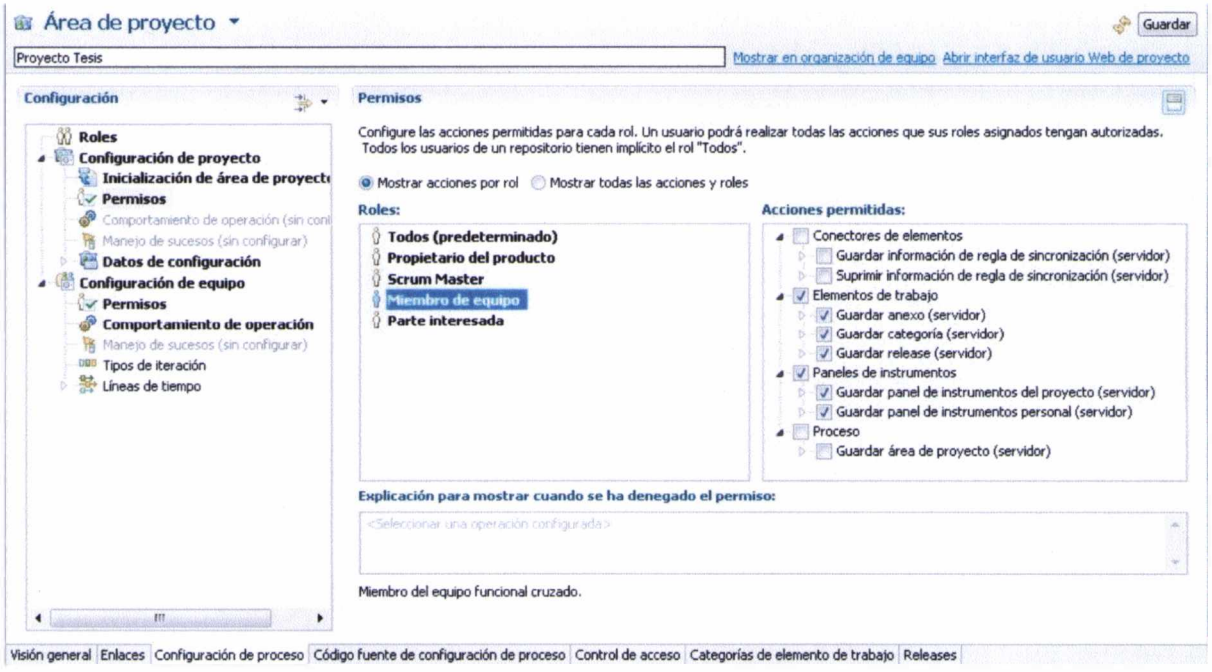

Figura 6.9. Configuración de Permisos

6.5.2. Creación y configuración del área de equipo

El paso siguiente que se optó por tomar (ya que todas estas tareas pueden realizarse en orden distinto al elegido) fue la creación, y posterior configuración del área de equipo.

La estructura de los equipos de proyecto se define por una o más áreas de equipo. Los proyectos complejos pueden presentar una jerarquía de áreas de equipo. Normalmente, uno o más equipos se asignan a cada línea de desarrollo. Los usuarios tienen múltiples asignaciones que requieren que trabajen en más de un equipo. Algunos miembros, como el jefe de proyecto, pueden no pertenecer a un área de equipo, pero se definen como miembros a nivel de proyecto en los aspectos generales del área de proyecto.

Aquí se definirá un nombre para el área de equipo y asociarlo a un área de proyecto. Opcionalmente se puede definir un resumen, un área de equipo padre, y una línea de tiempo donde tendrá incumbencia ese área de equipo.

En nuestro caso, como lo muestra la figura 6.10, se definió un nombre para el área de equipo y se lo asoció al área de proyecto creada con anterioridad.

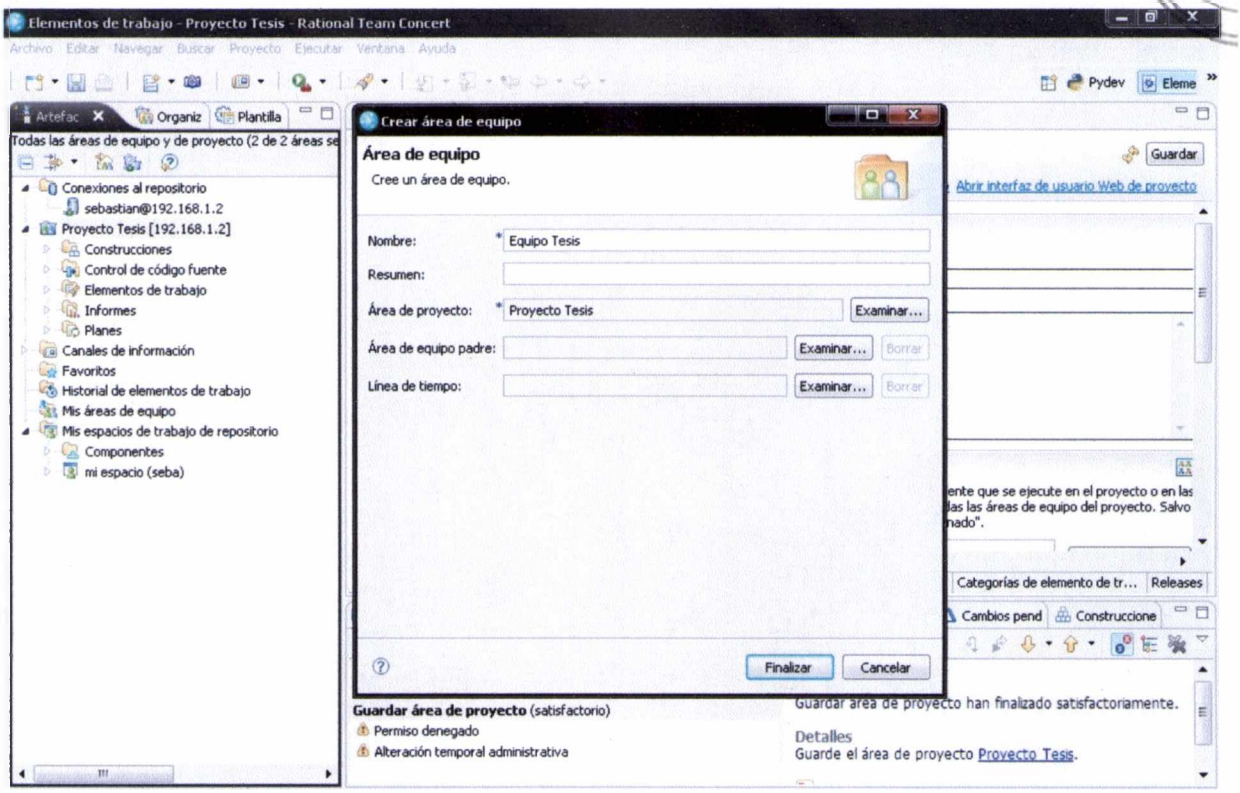

Figura 6.10. Creación de área de equipo

Hecho esto, se procede con la configuración del área de equipo.

Es necesario mencionar que resulta bastante confuso el hecho que un mismo usuario pueda tener distintos roles asignados en el área de proyecto y en el área de equipo (Por ejemplo, RTC permite que una misma persona pueda ser "Stakeholder" en el área de proyecto y "ScrumMaster" en el área de equipo). Pese a que en cada tipo de área (de proyecto y de equipo) se administran diferentes opciones, esta combinación de roles, según entendemos, supone la existencia de muchos mas roles que los definidos en Scrum.

En nuestro ejemplo lo primero que se realizó fue asignar los roles y los administradores. Se incorporaron 5 miembros al área de equipo con sus respectivos roles:

- Joaquín: Owner
- Lucrecia: ScrumMaster
- Sebastian y Francisco: Team Member
- Rodrigo: StakeHolder

En la figura 6.11 se muestra este paso.

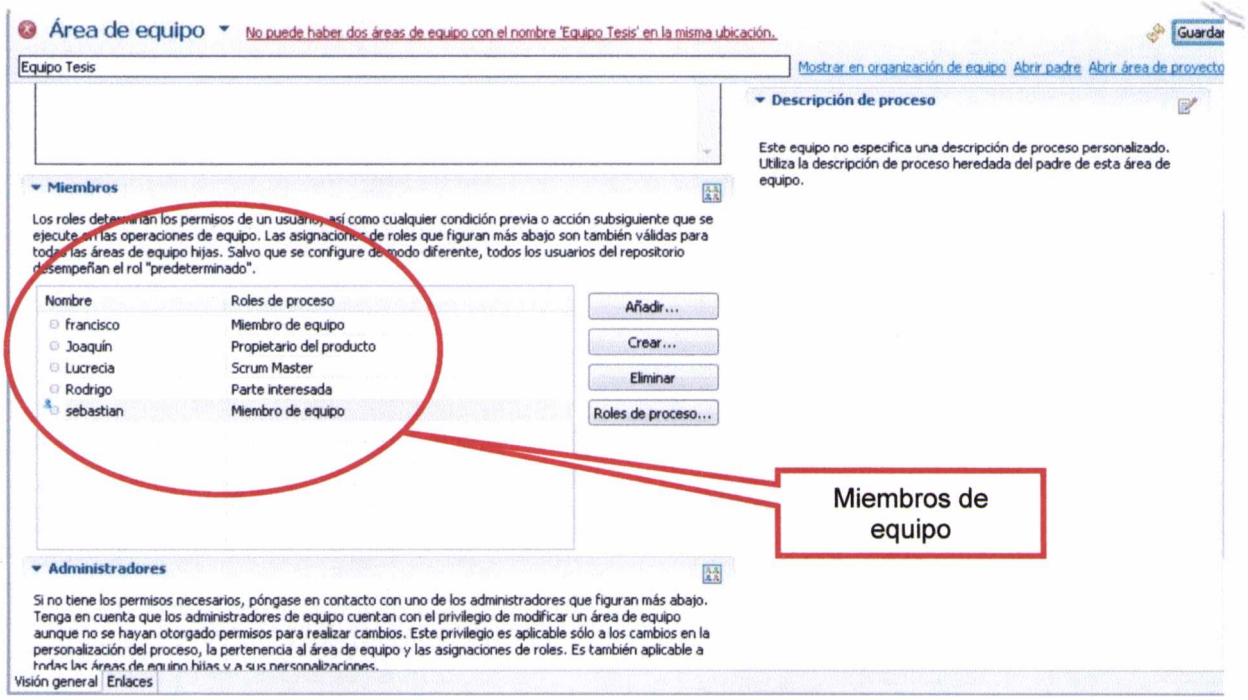

Figura 6.11. Configuración del área de equipo

6.5.3. Configuración de categorías de elementos de trabajo

Con el área de proyecto y el área de equipo creadas y configuradas, ahora se deben definir las categorías en las que se archivarán los elementos de trabajo y asociar las categorías a las áreas de equipo.

Esta característica es muy importante ya que se pueden tener diversas categorías, pudiendo asociar cada categoría a un área de proyecto o equipo.

Para lograr esto se debe ingresar nuevamente a la configuración del área de proyecto. En nuestro caso se definieron dos categorías, una para almacenar los elementos de trabajo del área de equipo y otra para guardar los elementos de trabajo del área de proyecto:

- Backlog (respetando la definición de Scrum): Categoría para el área de equipo
- Configuración: Categoría para el área de proyecto

En la figura 6.12. puede verse este paso.

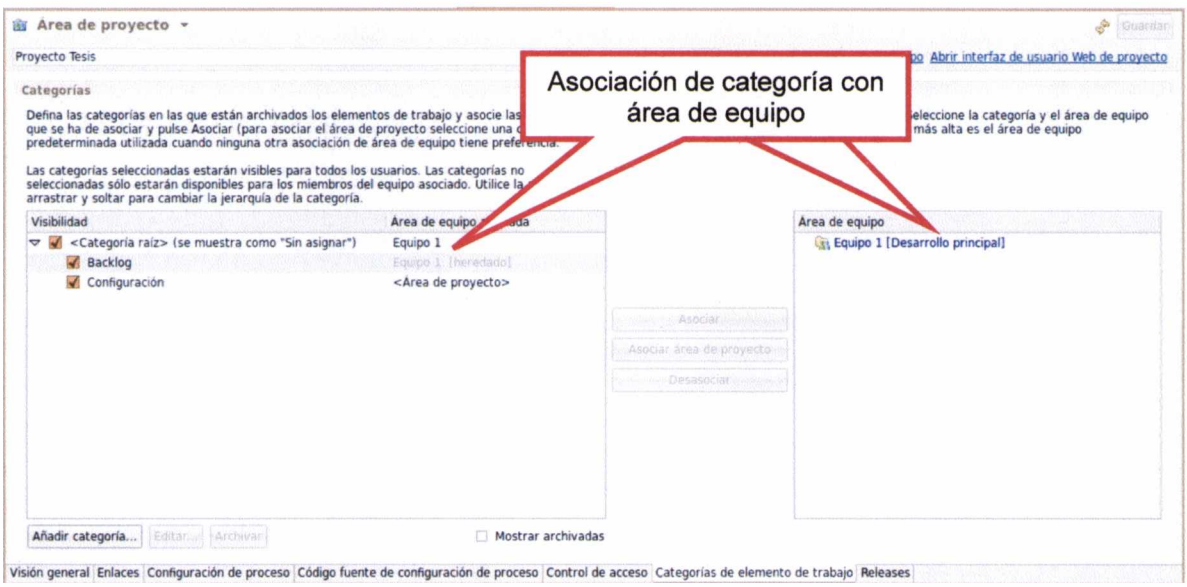

Figura 6.12. Definición de categorías de elementos de trabajo

6.5.4. Configuración de horas de trabajo

El paso a seguir, fue la definición de las horas de trabajo de los miembros del equipo. Ésta es una característica interesante de RTC ya que a partir de la asignación de la cantidad de horas de trabajo de los miembros y de los horarios de salida de cada miembro, el software calculará el rendimiento por cada usuario y el rendimiento general del equipo en el desarrollo de un proyecto.

Además, otra característica muy interesante, es que en el caso que un usuario forme parte de más de un área de proyecto y/o más de un área de equipo, RTC estima la distribución de la carga horaria para cada uno (siendo este valor modificable a gusto del usuario), para calcular luego a partir de esa estimación el rendimiento en el área respectiva.

En la figura 6.13. puede verse un ejemplo.

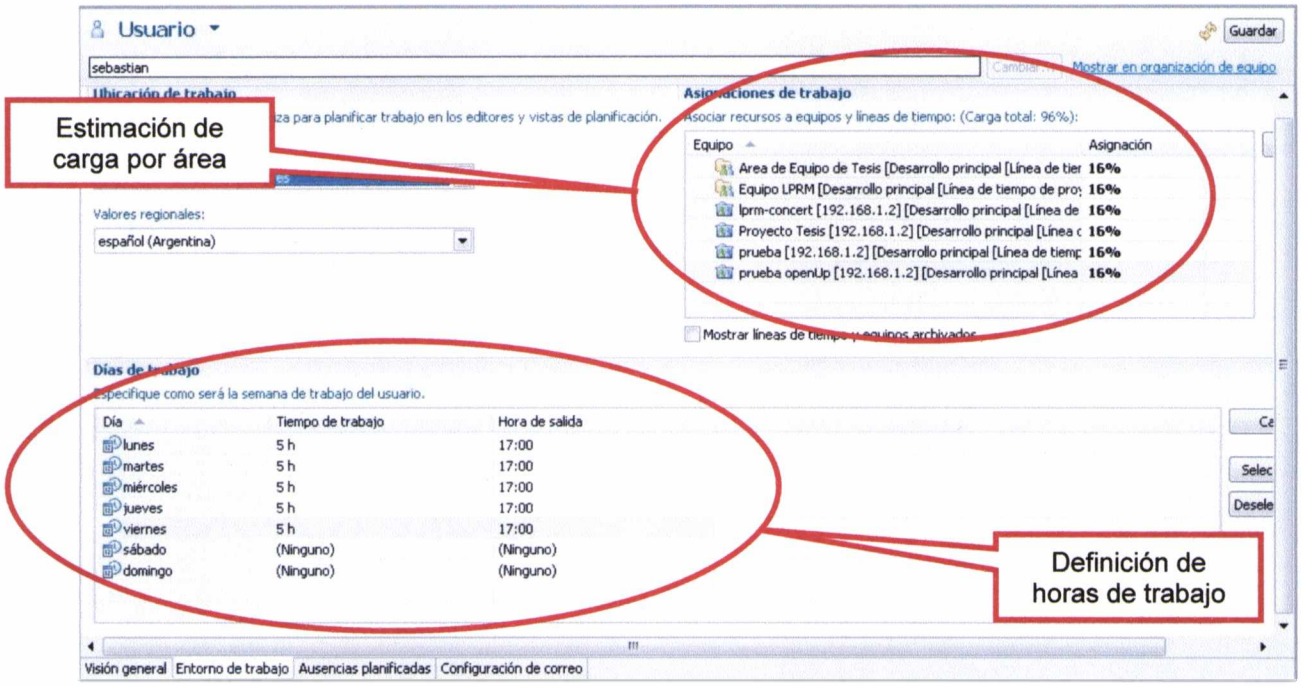

Figura 6.13. Definición de horas y estimación de carga por área

6.5.5. Configuración del plan

Como uno de los últimos pasos de configuración se debe proceder a la configuración de los planes.

Para los planes debe definirse el propietario que puede ser un área de proyecto o un área de equipo, y además, debe definirse la iteración que puede ser un release o un sprint determinado.

En nuestro caso, tenemos un único release y 2 sprint dentro del release, por lo que las iteraciones serán o el sprint 1 o el sprint 2.

Se definieron 2 planes para el sprint 1, uno para el área de proyecto y otro para el área de equipo, y un plan del sprint 2 para el área de equipo.

También se puede seleccionar un tipo de plan.

Los tipos de planes son una característica muy buena de RTC ya que determinan la forma en la que se visualizaran luego los Works ítems. Igualmente estas vistas se pueden configurar luego de creado el plan.

Otra característica importante es que se pueden crear varios planes por Sprint, lo cual presupone que pueden definirse varios planes para distintas áreas de equipo.

En la figura 6.14. se muestra el ejemplo de creación de un plan.

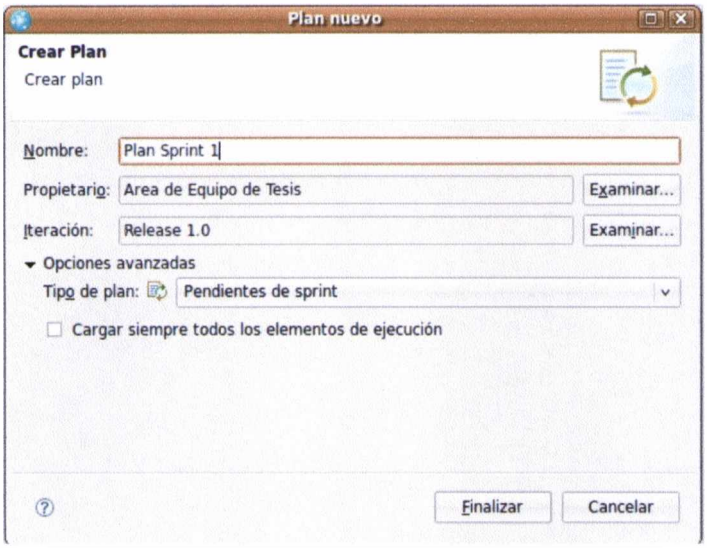

Figura 6.14. Creación de un plan

6.5.6. Creación y configuración de elementos de trabajo

Respecto a configuraciones, este fue el anteúltimo paso que se realizo.

Los elementos de trabajo son fundamentales en RTC, ya que la mayoría de la lógica del soft en la plantilla Scrum funciona en base a ellos.

Los elementos de trabajo existentes son:

- *Historia*: define lo que es necesario construir. Las historias se estiman en puntos de historia y se desglosan en tareas.
- *Epic*: se utiliza cuando una historia es demasiado grande para completarse en un solo sprint o cuando hay muchas incertidumbres para estimar la cantidad de trabajo. Un Epic se puede dividir en varias historias.
- *Tarea*: define una unidad de trabajo planificada para un sprint y se estima en horas. Las tareas se vinculan a una historia mediante un enlace padre/hijo.
- *Defecto*: un error, un fallo o una imperfección de la implementación. Los defectos relacionados con una historia se pueden asociar mediante un enlace padre/hijo. Los defectos y las tareas asociados con la historia se muestran en el panel de tareas.
- *Impedimento*: se utiliza para realizar seguimiento de los problemas que impiden avanzar.
- *Retrospectiva*: se utiliza para capturar las conclusiones de la reunión retrospectiva de sprint.
- *Seguimiento de elemento de construcción*: por lo general, se crean a partir de un resultado de construcción para realizar seguimiento de los arreglos necesarios para una construcción con errores.
- *Elemento de adopción*: realiza seguimiento del momento en que es necesario que un equipo adopte los cambios realizados por otro.

En nuestro caso sólo se trató con los elementos "Historia", "Tarea" y "Retrospectiva".

Para cada elemento de trabajo se debe indicar:

- En que categoría se archivará
- Una descripción
- Un código
- El área de equipo y el área de proyecto a la cual pertenece

- De quien es propiedad ese elemento
- La prioridad de ese elemento
- Para que iteración o release esta planificado
- La estimación es horas que demandara concluirlo

Al mismo tiempo quedaran registrada información tal como:

- Fecha de creación
- Quien lo creo

Y otra información que quedará registrada y/o puede agregarse como:

- Corrección
- Tiempo restante
- Fecha de vencimiento
- Fecha de resolución
- Quien lo resolvió
- Suscripciones
- Etc.

La definición formal de Scrum sugiere que todas las tareas se registran en un Backlog y que cada miembro de equipo se va asignando esas tareas. Si se considera esa definición, RTC no respetaría ese principio ya que permite que un usuario asigne tareas a otros y, hasta incluso, permite a usuarios concluir tareas ajenas (Puede configurarse, pero por defecto esta definido así)

RTC es muy flexible y configurable... Sin embargo, este factor influye negativamente en algunos aspectos. Hay muchos elementos o características que tienen representación pero no funcionalidad, por ejemplo, la dependencia o la relación de tareas, o el elemento de trabajo "Retrospectiva".

En la figura 6.15 se muestra el ejemplo de un elemento de trabajo (tarea).

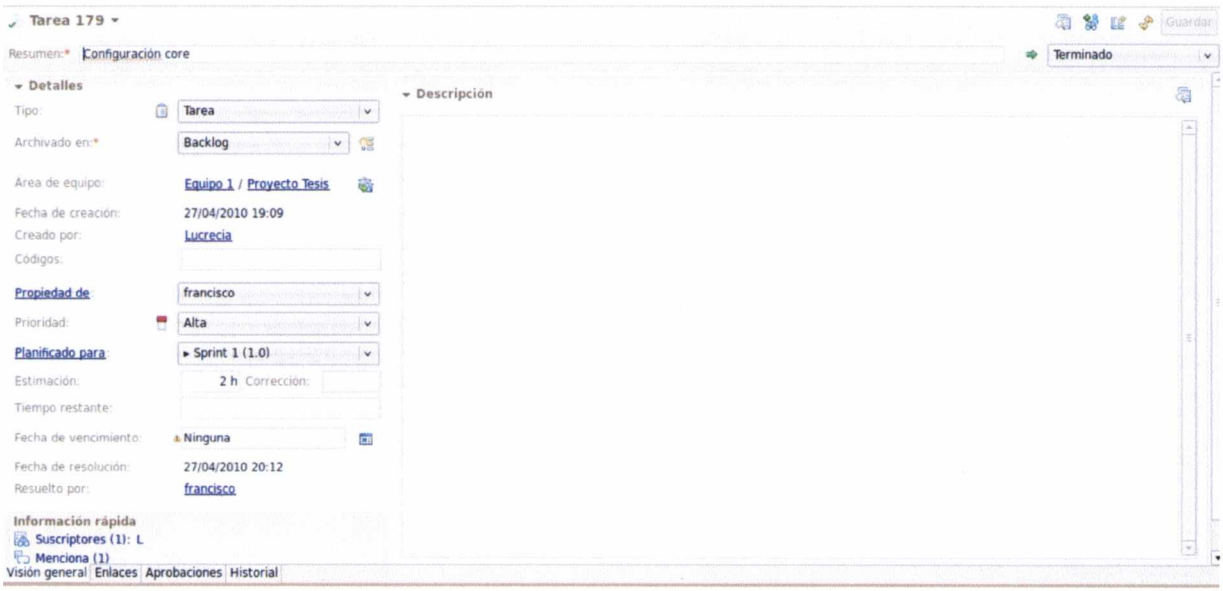

Figura 6.15. Ejemplo de un elemento de trabajo

6.5.7. Configuraciones adicionales realizadas

Por último se configuraron algunas reglas necesarias para el proyecto propuesto. Se configuraron reglas para la entrega de código fuente. Nos pareció importante mencionar algunos detalles en este caso, ya que observamos características interesantes como que es posible entregar código con o sin errores, se puede configurar para "obligar" a los miembros a relacionar el código con una tarea, o a entregar el código con comentarios adjuntos. En la figura 6.16 se muestra el caso en el que se eliminó una regla de entrega de código fuente para permitir a los miembros entregar código fuente, aun si falla en compilación.

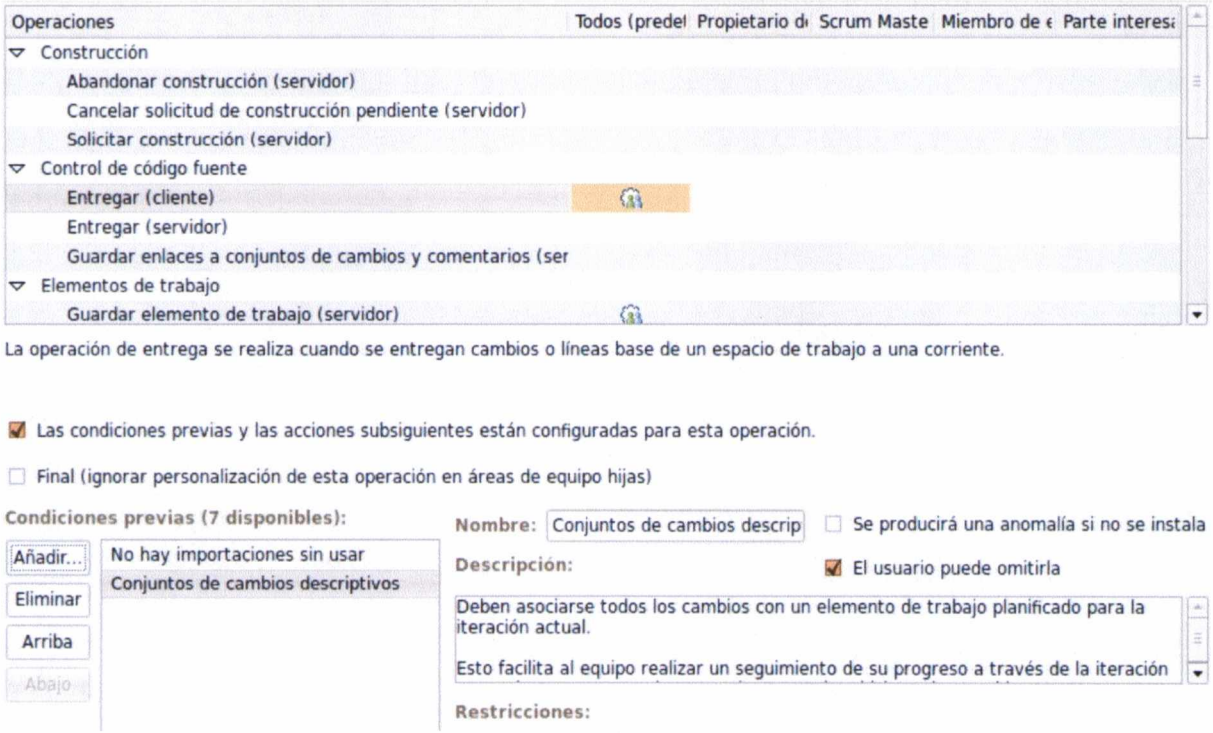

Figura 6.16. Configuración de reglas de entrega de código fuente

6.5.8. Ejecución: Respecto a elementos de trabajo

Terminada la configuración y llegada la fecha de inicio del primer sprint comenzó la ejecución del proceso.

En lo que concierne a la ejecución, se explicarán algunos detalles encontrados que se creen los mas relevantes para contar y algunos detalles o curiosidades que nos llamaron la atención.

Ya no se explicará paso a paso lo que se hizo, si no que se dividirá el contenido en ítems.

En este apartado se mostrarán algunos screenshot respecto al comportamiento de los elementos de trabajo en ejecución.

Una característica importante que se pueden encontrar en los Work Items es que estos guardan todo el historial de acciones que se ejercen sobre ellos: Quién lo creo, quién lo inició, quien cambio determinado estado, código fuente asociado a ellos, si se adjuntó algún enlace (archivo, dependencia, relación), etc. En la figura 6.17. puede verse el ejemplo del historial de un elemento de trabajo.

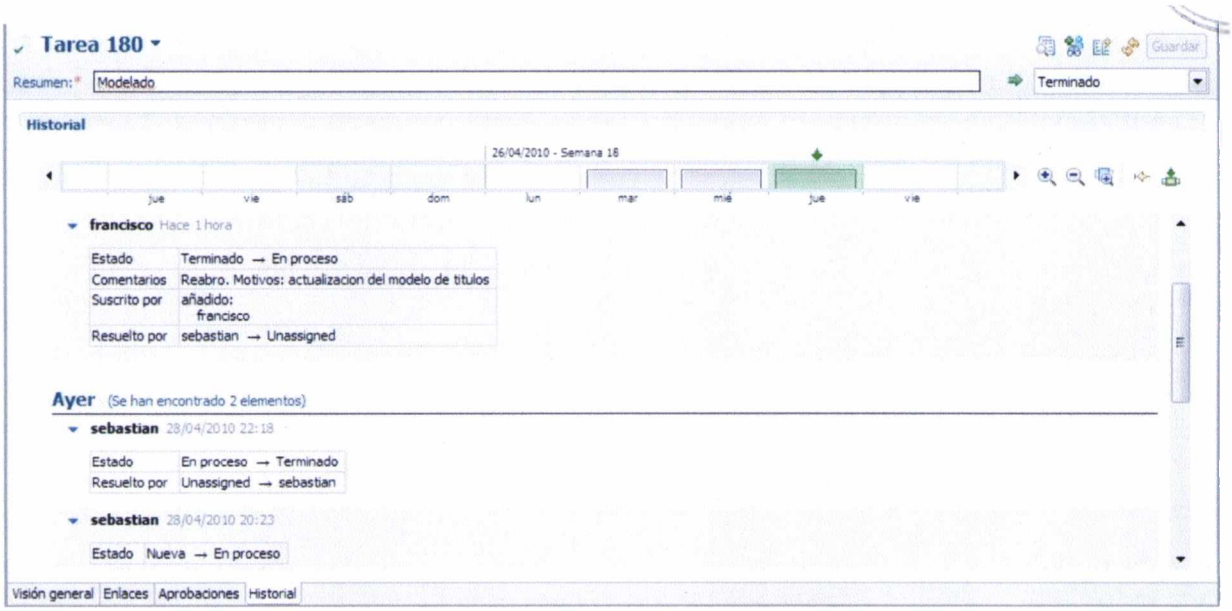

Figura 6.17. Historial de un elemento de trabajo

Otra funcionalidad de RTC que nos pareció interesante mostrar, es que se puede visualizar el desarrollo de los elementos de trabajo de cada miembro por día, incluyendo los elementos que aun quedan por completar. RTC sugiere qué tareas completar en los días siguientes, teniendo en cuenta la carga horaria y la prioridad de cada elemento de trabajo, y las horas de trabajo del miembro de equipo. En la figura 6.18 puede verse un ejemplo.

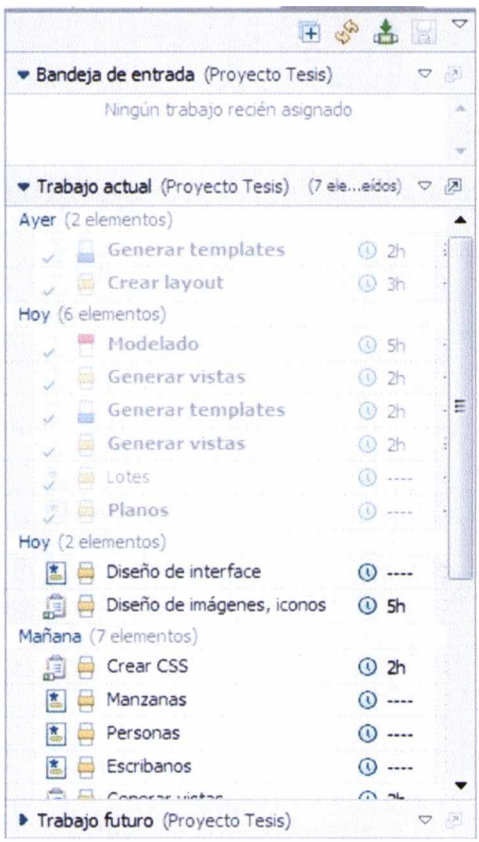

Figura 6.18. Distribución de tareas por día

Se muestra además el ejemplo de una Retrospectiva concluida, con una discusión y observaciones. Éste es un ejemplo de la gran representación que ofrece RTC, pero al mismo tiempo, de la poca funcionalidad, ya que este tipo de elemento no ofrece prácticamente nada diferente a los otros elementos de trabajo. Este ejemplo puede verse en la figura 6.19.

Figura 6.19. Ejemplo de Retrospectiva

Por ultimo se muestra una interesante herramienta que presta RTC. Se trata de un generador de Screenshot incluido en el soft. Desde cada elemento de trabajo puede generarse una instantánea y adjuntarla al mismo elemento, enviársela a un miembro, guardarla en el equipo personal, etc. En la figura 6.20 puede verse este ejemplo.

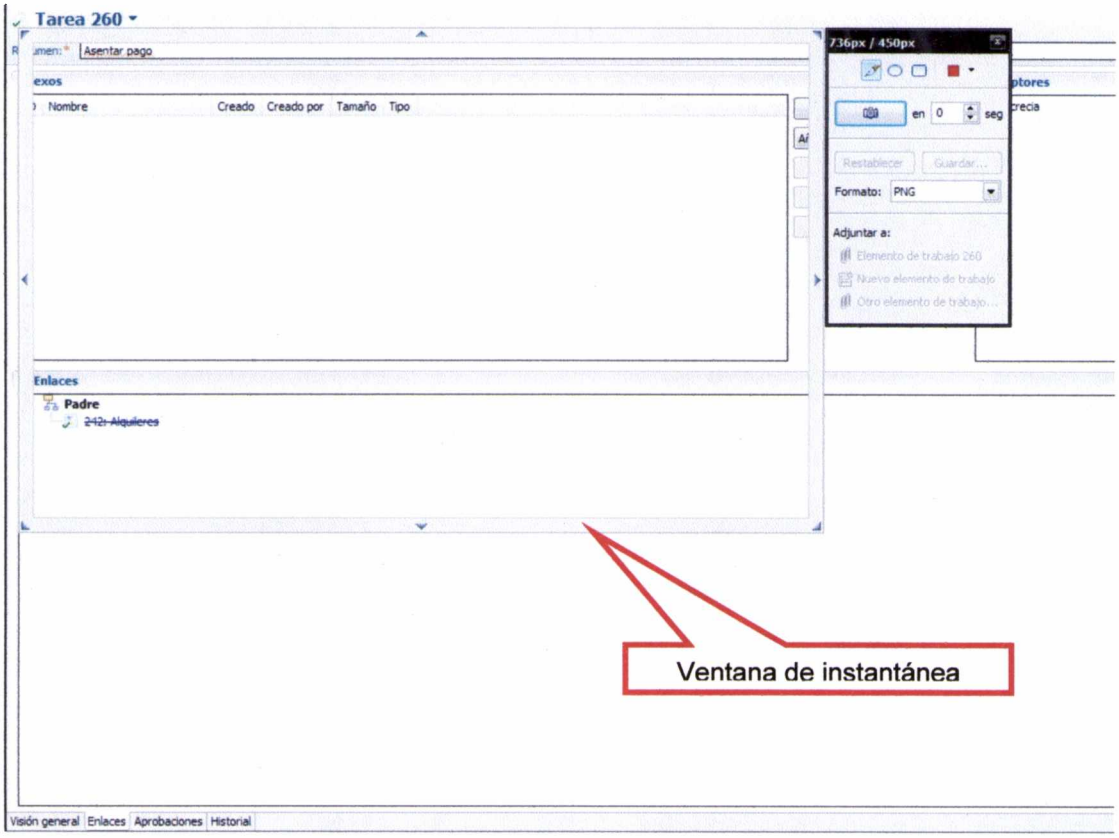

Figura 6.20. Instantánea desde elemento de trabajo

6.5.9. Ejecución: Respecto a los Sprint, los planes y las consultas

Ya se mostraron algunos ejemplos sobre las tareas. En este apartado se mostrarán algunos ejemplos sobre el sprint y los planes en general.

Como ya se sabe, existen dos tipos de planes en un sprint, el plan de proyecto y el plan de equipo.

Los planes tienen importantes características que hacen a RTC, en este aspecto, un software muy productivo y muy útil.

Lo primero a remarcar es que RTC va monitoreando el progreso general del equipo y el progreso particular de cada miembro en base a las horas estipuladas de los elementos de trabajo, en base a lo que se tarda en realidad en concluir una tarea, en base al estado de cada elemento de trabajo y en base al cronograma de horas de trabajo que configuró cada miembro. Por ejemplo, este aspecto puede ayudar a cada miembro a refinar la estipulación de horas de cada elemento de trabajo para un próximo Sprint en caso de que la diferencia entre las horas estipuladas y la horas reales sea considerable; Ayuda también a los miembros a conocer el estado de su progreso, para, en base a esa información, saber si está cumpliendo con la velocidad y eficiencia esperada y actuar acorde a eso; Ayuda al Owner a conocer cual es el estado de su producto; Ayuda al ScrumMaster a conocer el desempeño del equipo y actuar acorde a eso, etc.

Otro aspecto, no menos importante, es que permite obtener la vista deseada por cada miembro acerca del progreso del plan (Carpetas de equipo, División de trabajo, Lista de rangos, panel de tareas de desarrollador, tiempo planificado), pudiendo filtrar los elementos que se deseen (Elementos asignados, elementos en ejecución, elementos estimados, elementos no modificados, elementos resueltos, grupos vacíos). Este es otro factor de importancia al momento de remarcar la flexibilidad de RTC.

En nuestro caso, solo hay una tarea en el plan de proyecto que responde al ScrumMaster, quién se encargó de todos los procesos de configuración. Se muestra el ejemplo de cuando se inició esa tarea en la figura 6.21.

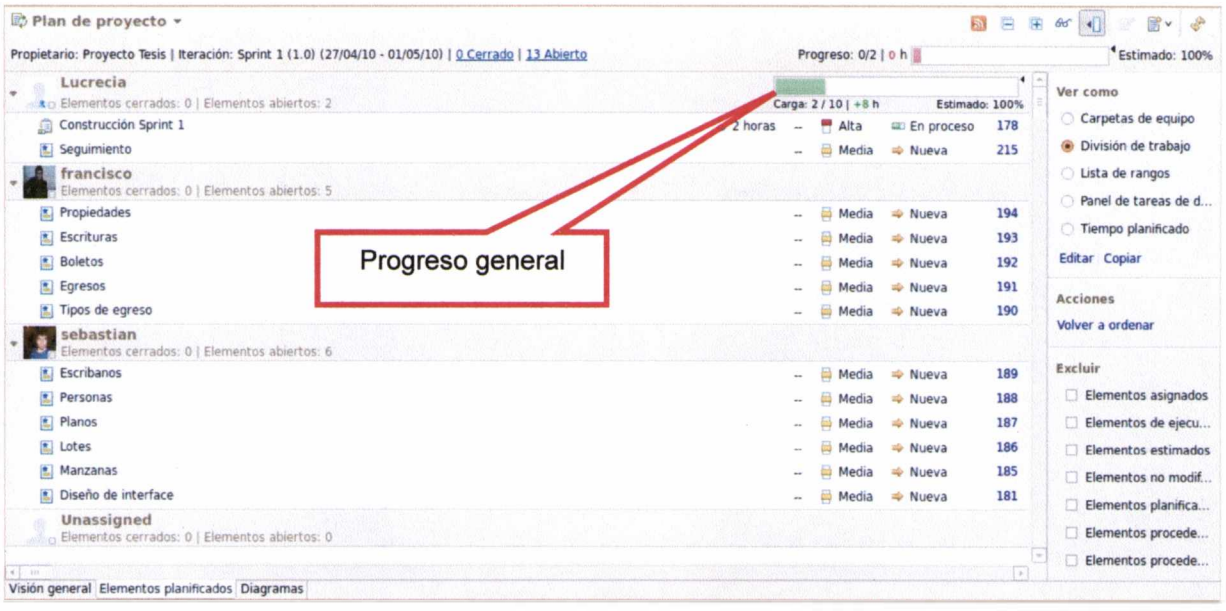

Figura 6.21. Plan de Proyecto

El plan de equipo es más interesante, ya que casi todos los miembros tienen asignadas varias tareas. Este ejemplo puede verse en la figura 6.22.

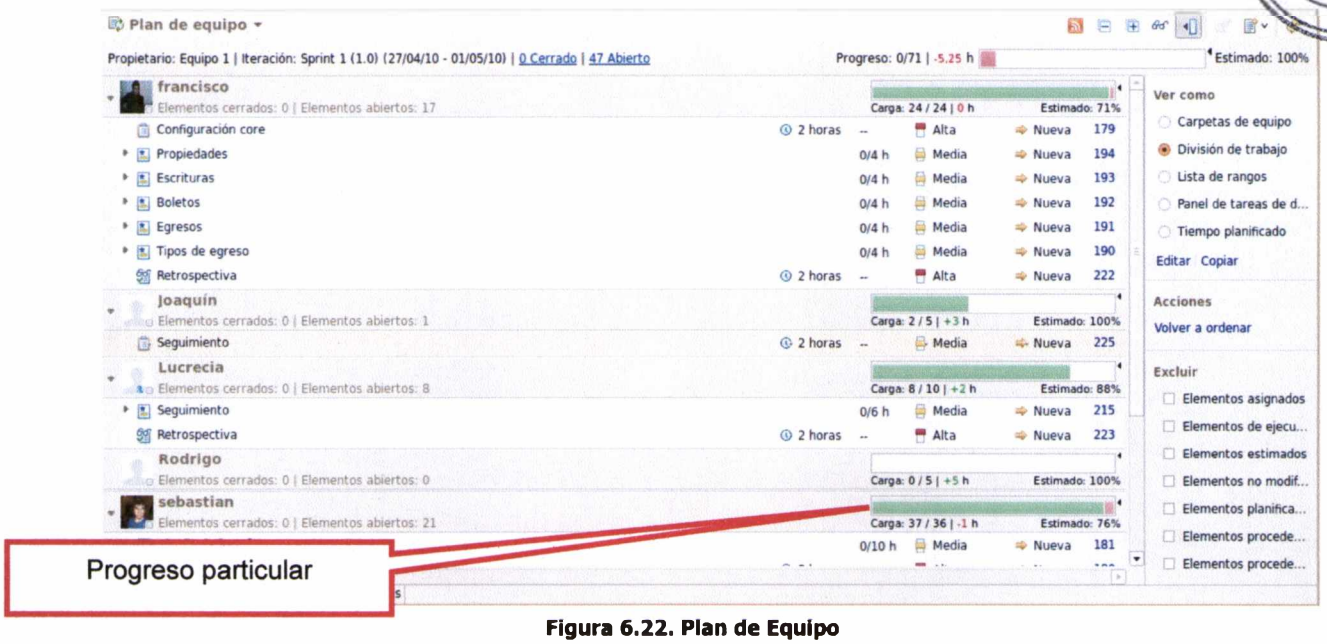

Figura 6.22. Plan de Equipo

Y en la figura 6.23. puede verse el plan de equipo luego de finalizado el Sprint.

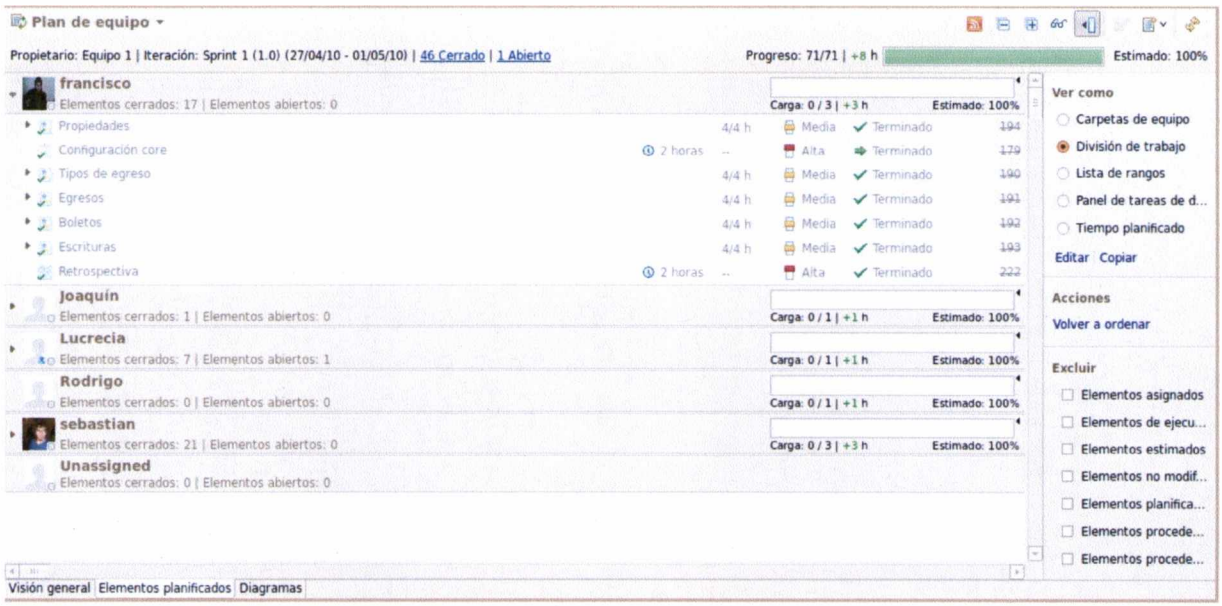

Figura 6.23. Plan de Equipo finalizado

Además en este apartado quisimos exponer un conflicto con el cual nos encontramos: Iniciamos sesión con los usuarios "Sebastian" y "Francisco" respectivamente (los dos con el rol Team Member) e hicimos acceder a ambos a una misma tarea. Con el usuario "Sebastian" se modificó el estado de la tarea y se guardaron los cambios. Mientras con el usuario "Francisco", se modificaron los puntos de historia; Cuando se intentaron guardar los cambios se presento una advertencia que sugería fusionar o descartar cambios ya que la tarea había sido modificada. Se probaron ambas opciones. En caso de descartar, se descarga la tarea con los datos actualizados, y el miembro debe volver a implementar los cambios. En caso de fusionarse, siempre y cuando no se estén modificando los mismos campos, permite aceptar ambos cambios. En nuestro caso se trató de esto último de modo tal que nos permitió fusionar los cambios. En la figura 6.24. Se muestra este conflicto.

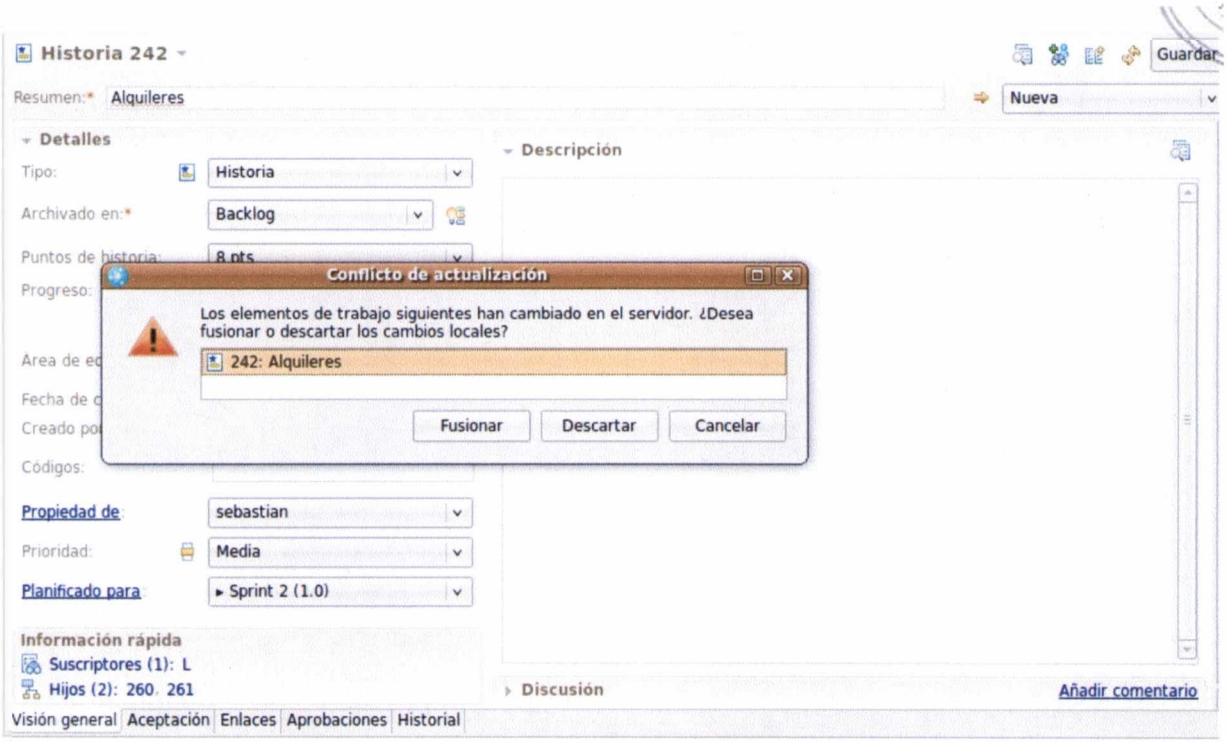

Figura 6.24. Conflicto en plan

Uno de los aspectos a remarcar es que por definición de Scrum "Las tareas en el *sprint backlog* nunca son asignadas, son tomadas por los miembros del equipo del modo que les parezca oportuno.". En la plantilla Scrum por defecto, RTC permite que los usuarios asignen tareas a otros usuarios (Por ejemplo el ScrumMaster puede asignarles las tareas a los TeamMember). Es entendible que, tratándose de un software de éstas características, se permita realizar esto... pero nos pareció interesante mencionarlo.

Otro de los aspectos a objetar, es que por definición Scrum sugiere que "No se puede cambiar los objetivos/requisitos de la iteración en curso". Pese a que es entendible que pueden ocurrir fallas menores tales como de tipeo, RTC permite en medio de una iteración agregar tareas, eliminar tareas, cambiar de Sprint a las tareas, etc.

Respecto a los planes y los sprint queda por mencionar un problema muy importante con el cual nos encontramos que creemos que atenta seriamente contra la fiabilidad del software: En nuestro caso, se creo un plan de proyecto y un plan de equipo para el Sprint 1. A modo de prueba, luego de armar los planes, asignar tareas, estipular horas, puntos de historia, etc., decidimos modificar la fecha de inicio del Sprint. Luego de hacer esto sucedió que los dos planes estaban vacíos. Si bien las tareas quedaron almacenadas en un repositorio, se desarmaron los planes. Nos parece muy importante remarcar esto, porque por cada Sprint se pueden crear innumerables cantidad de planes, asignados a diferentes áreas de equipos, y si esto sucediese en un grupo de trabajos realmente grande donde se debe desarrollar un proyecto grande, afectaría directamente el desarrollo completo del proyecto.

6.5.9.1. Consultas en los planes

Otra herramienta muy útil que facilita la búsqueda de elementos de trabajo mediante una considerable cantidad de filtros, son las consultas. Existen muchas consultas predefinidas que son las más utilizadas. Sin embargo también se pueden crear consultas personalizadas lo cual vuelve a potenciar la flexibilidad de la herramienta.

En la figura 6.25. se muestra el ejemplo de una consulta predefinida.

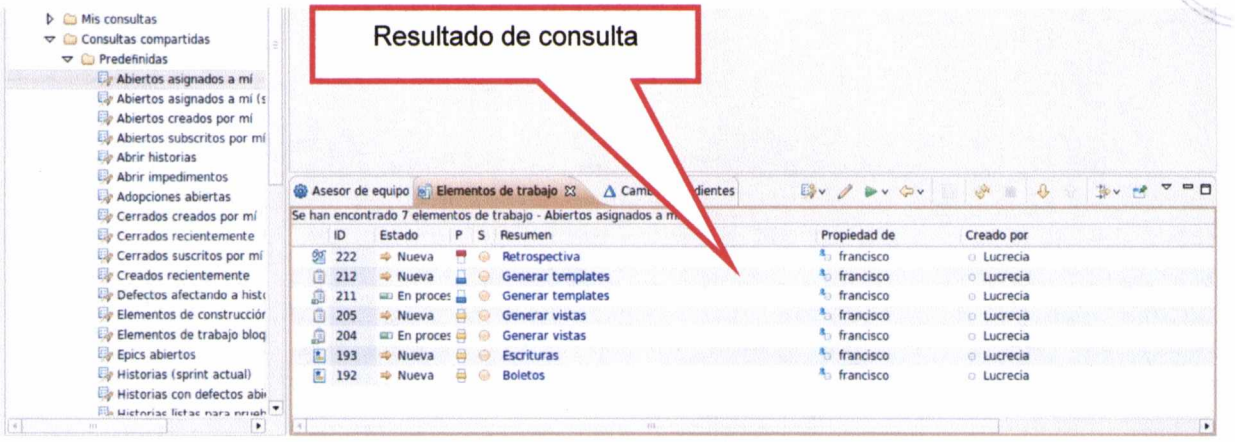

Figura 6.25. Consulta predefinida

En la figura 6.26. se muestra el ejemplo de una consulta personalizada.

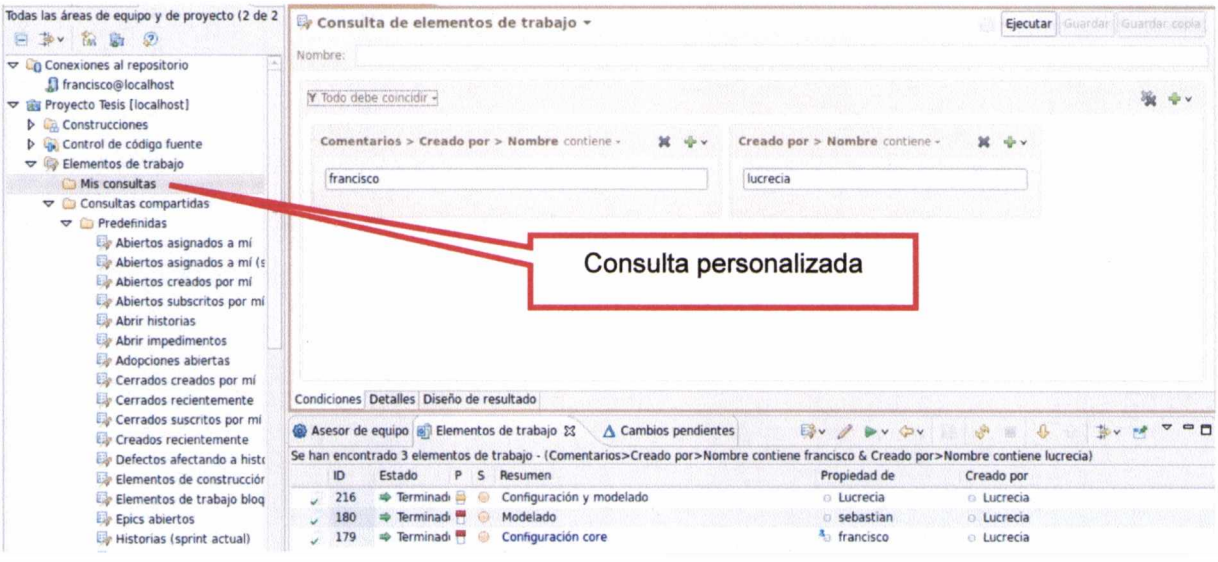

Figura 6.26. Consulta personalizada

6.5.10. Ejecución: Respecto a la administración de código fuente

RTC además de ser un administrador de procesos, es un administrador de código fuente.

En este apartado se expondrán características encontradas en este aspecto.

El sistema de control de versiones de RTC en principio, no aporta demasiadas diferencias a otros ya conocidos. Sin embargo aquí se contará la experiencia de utilizarlo.

La administración del proyecto se visualiza y ordena como cualquier proyecto desarrollado en eclipse, lo cual implica una gran ventaja para aquellas personas que conocen el IDE.

Lo primero a exponer, el caso más simple, es cuando un usuario quiere entregar los archivos que generó al repositorio principal. Cuando se generan cambios en el repositorio local (puede ser cualquier archivo, no solo código fuente), RTC alerta al usuario advirtiéndole que hay información "Saliente" lista para entregarse, y a qué tarea pertenece esa información (en caso de haber configurado que toda entrega de archivos debe estar asociada a un elemento de trabajo, siempre debe explicitarse esta asociación... pero puede no hacerse). Este ejemplo puede verse en la figura 6.27.

Figura 6.27. Código saliente

Una vez entregado el código sin conflictos previos los involucrados en el proyecto y en la administración de código fuente serán alertados en su terminal en la solapa "cambios pendientes" sobre la existencia de información "Entrante" como puede verse en la figura 6.28.

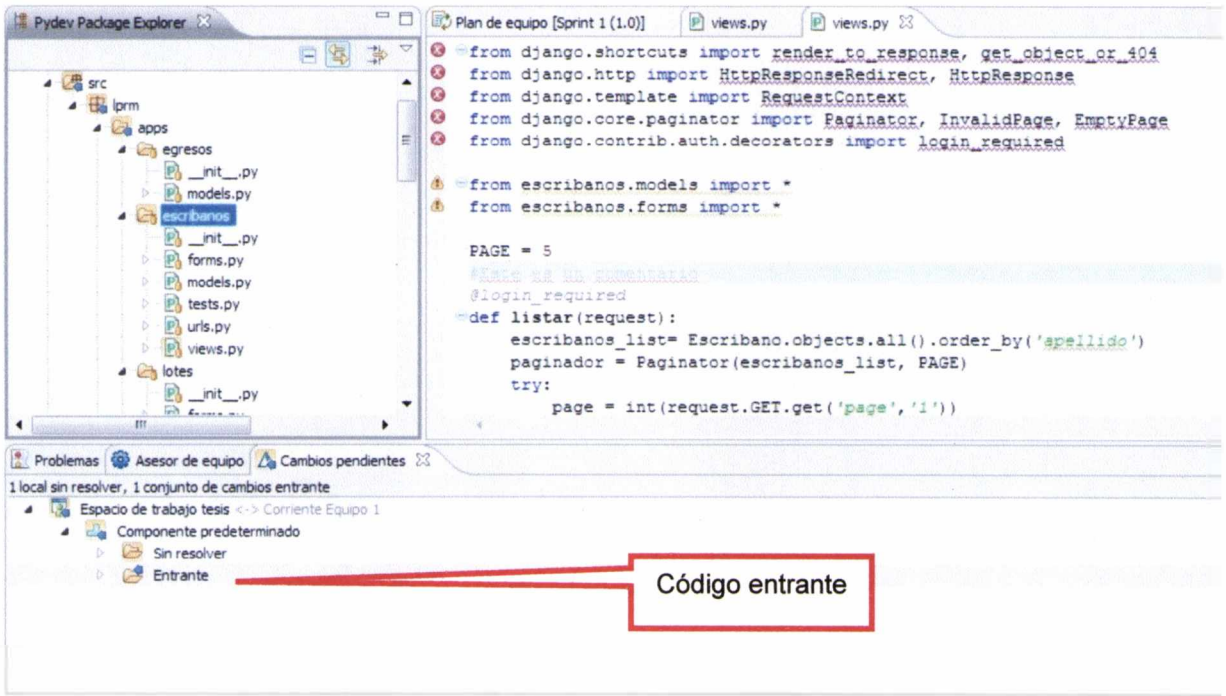

Figura 6.28. Código entrante

El manejo de versiones de código de RTC, se ve en forma explícita cuando dos o mas personas tienen en su repositorio local una misma versión de un archivo, y luego de que uno de los miembros entrega su versión, los otros intentan hacer lo mismo. En esa situación se evidencia un conflicto de versiones, a la que RTC propone dos soluciones:

- Una resolución automática: Que se fusionen dos archivos automáticamente significa que pueden convivir las modificaciones hechas por dos usuarios. Para que la fusión automática pueda efectivizarse, se tiene que dar el caso en el que los miembros hayan modificado diferentes bloques de código (por ejemplo, hablando de un código

fuente de 200 líneas, que un usuario haya modificado la línea 20 hasta la 40 y el otro la línea 150 hasta la 200). En caso que puedan fusionarse se incluyen los archivos fusionados en la carpeta "Saliente" y se debe entregar el código. Si no se puede realizar la fusión automática deberá optarse por una fusión manual.

- Una resolución manual: Siempre es posible solucionar un conflicto con esta fusión. En la fusión manual se muestra al usuario la versión de los dos archivos en conflicto indicándole las diferencias existentes entre los dos. Es responsable el usuario de realizar una fusión en forma correcta.

En el ejemplo de las figura 6.29. se muestra cuando RTC alerta sobre la existencia de cambios que faltan reincorporar y que pueden causar conflictos, antes de entregar.

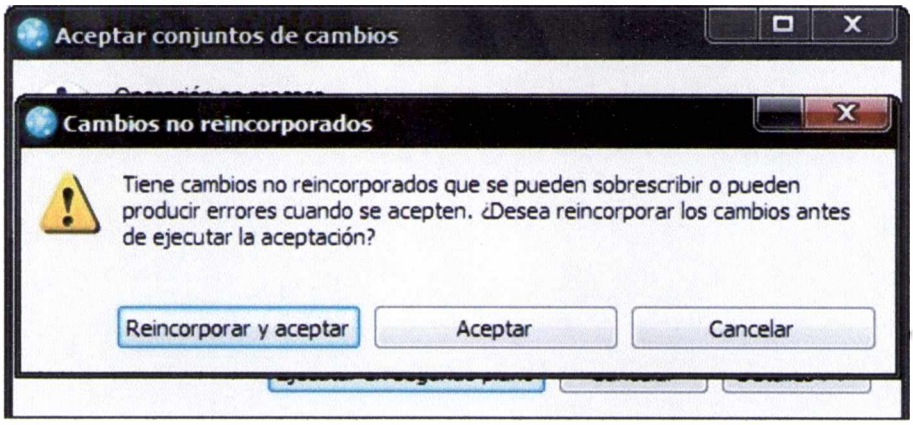

Figura 6.29. Advertencia sobre posible conflicto

Confirmado el conflicto RTC alerta al usuario. Estos conflictos se registran en la solapa "Cambios pendientes" como "Sin Resolver". Como lo muestra la figura 6.30.

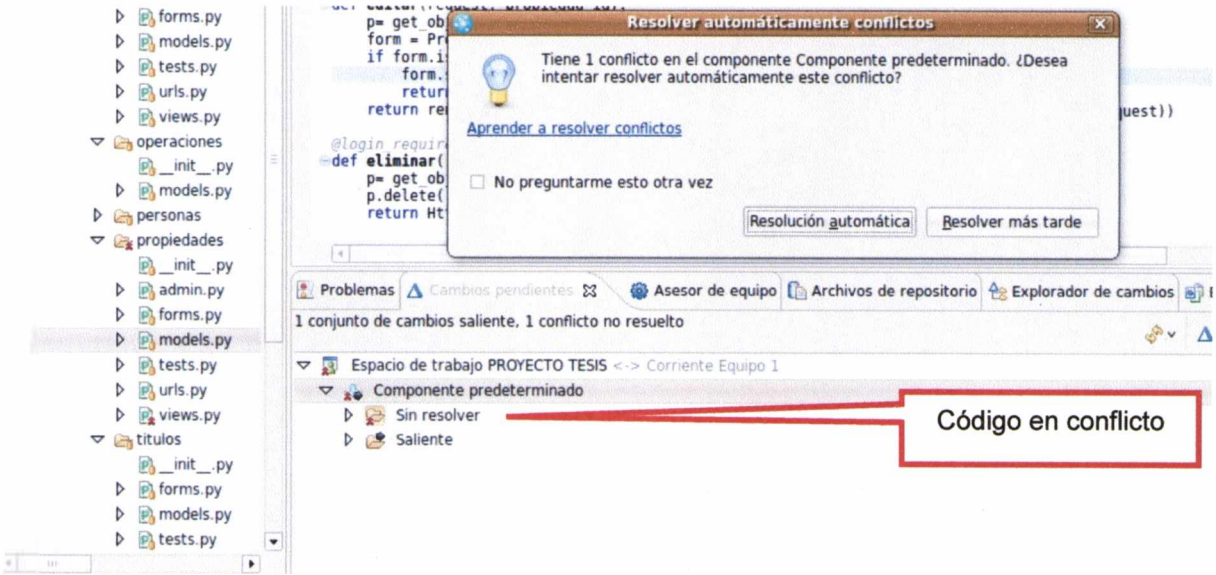

Figura 6.30. Conflicto de código

La resolución automática simplemente fusiona los archivos en uno y lo coloca como información "Saliente". En nuestro caso queremos mostrar los pasos a seguir de dos archivos que no se pudieron fusionar automáticamente, forzando así una fusión manual.

Luego de intentar fusionar automáticamente RTC advirtió con el error de la figura 6.31 que no podía realizar una fusión automática.

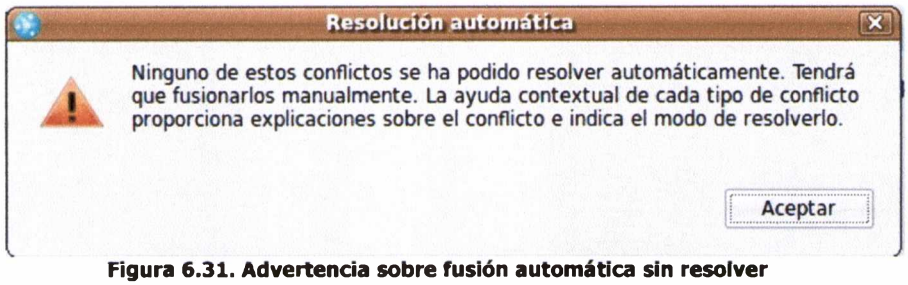

Figura 6.31. Advertencia sobre fusión automática sin resolver

Entonces se procede con la fusión manual donde RTC presenta un visor con los dos archivos en conflicto para que el usuario realice las modificaciones manualmente, como lo muestra la figura 6.32.

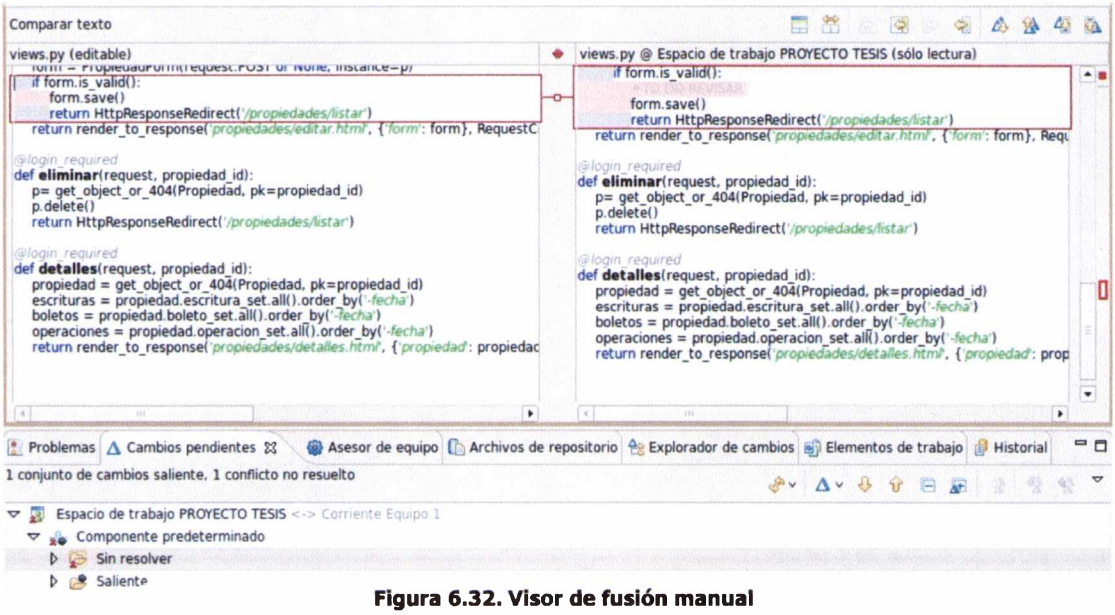

Figura 6.32. Visor de fusión manual

Y luego de realizados los cambios RTC nos advierte sobre que se reescribirá el archivo con el cual surgió el conflicto, como lo muestra la figura 6.33.

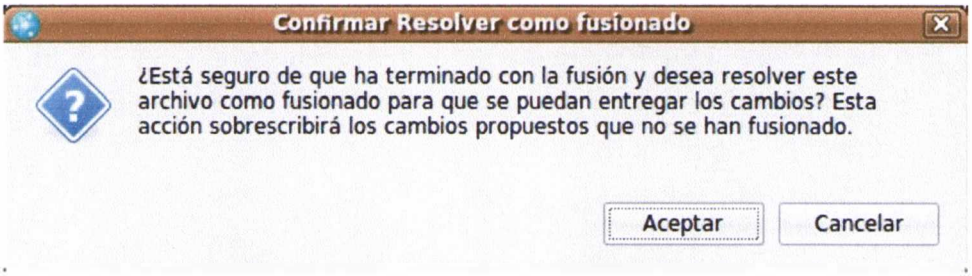

Figura 6.33. Advertencia sobre fusión manual

Y entonces sí, se fusionan los archivos como se puede ver en la figura 6.34.

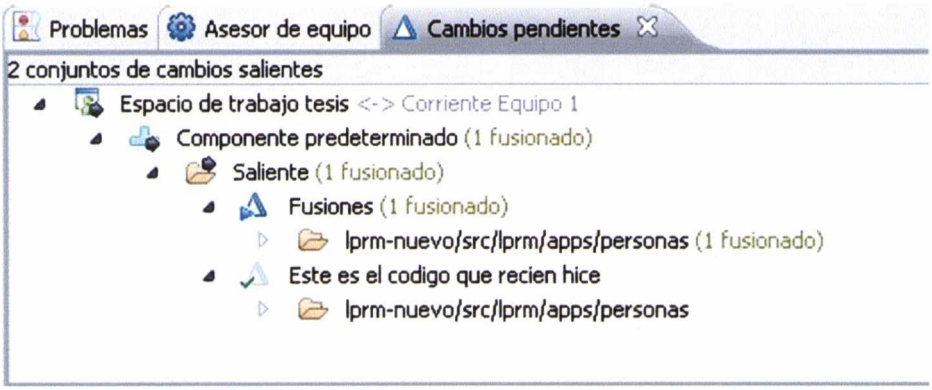

Figura 6.34. Fusión concluida

Como últimos ejemplos, quisimos presentar otro tipo de errores que surgieron cuando existían conflictos.

En la figura 6.35 se muestra un error de conexión cuando se intentaba completar una fusión.

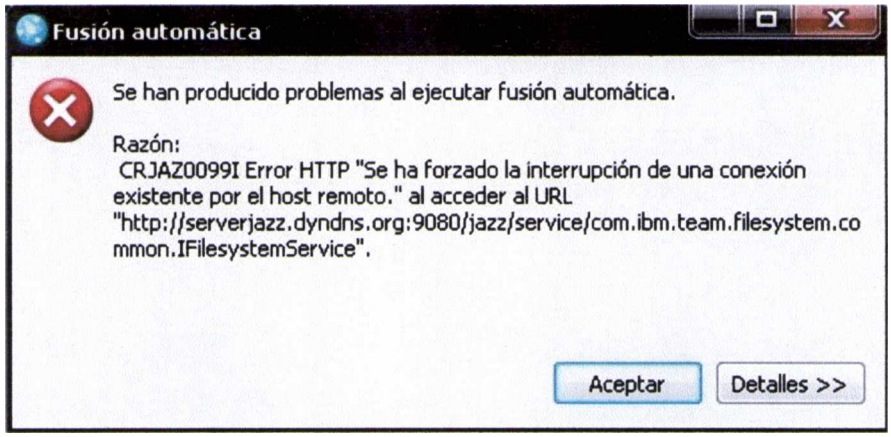

Figura 6.35. Error de conexión

En la figura 6.36. se muestra un error cuando se intento reincorporar y entregar código con conflictos pendientes. Aquí es interesante aclarar que cuando se quisieron obtener mas detalles del error, se obtuvo el mismo mensaje de error.

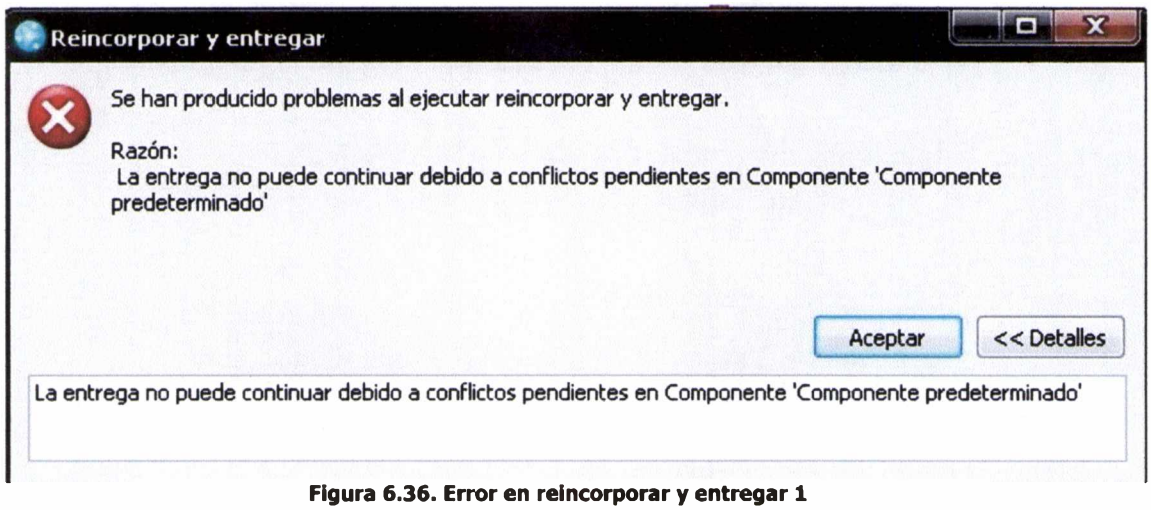

Figura 6.36. Error en reincorporar y entregar 1

Y el la figura 6.37 se presento un error cuando también se intento reincorporar y entregar código que produciría conflicto en el repositorio local.

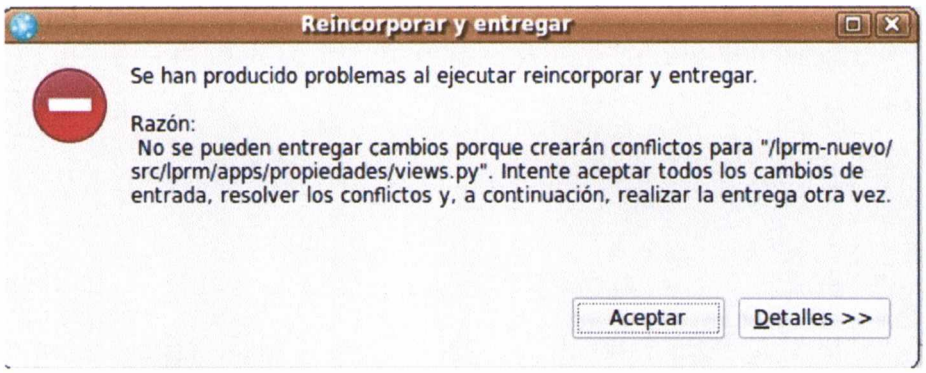

Figura 6.37. Error en reincorporar y entregar 2

6.5.11. Ejecución: Respecto a la comunicación

RTC brinda, además, distintas formas de comunicación que nuevamente hacen al software muy flexible.

Existe la comunicación entre miembros directa e indirecta. Las formas directas son las suscripciones a las tareas o a las personas, colaboración en contexto (Ej. discusiones en los elementos de trabajo), la posibilidad de conocer si está en línea un miembro determinado, etc. Las formas indirectas son las historias de los elementos de trabajo, los cambios realizados en la corriente de equipo, el registro de eventos, etc.

De este modo cualquier miembro puede configurar a su gusto y recibir la información que desee para estar al tanto de los cambios surgidos en el proyecto

Como ejemplo se expondrán algunos de estos casos.

Se muestra el ejemplo en la figura 6.38. de una "conversación" en una tarea iniciada por el usuario "Francisco" hacia el usuario "Sebastian".

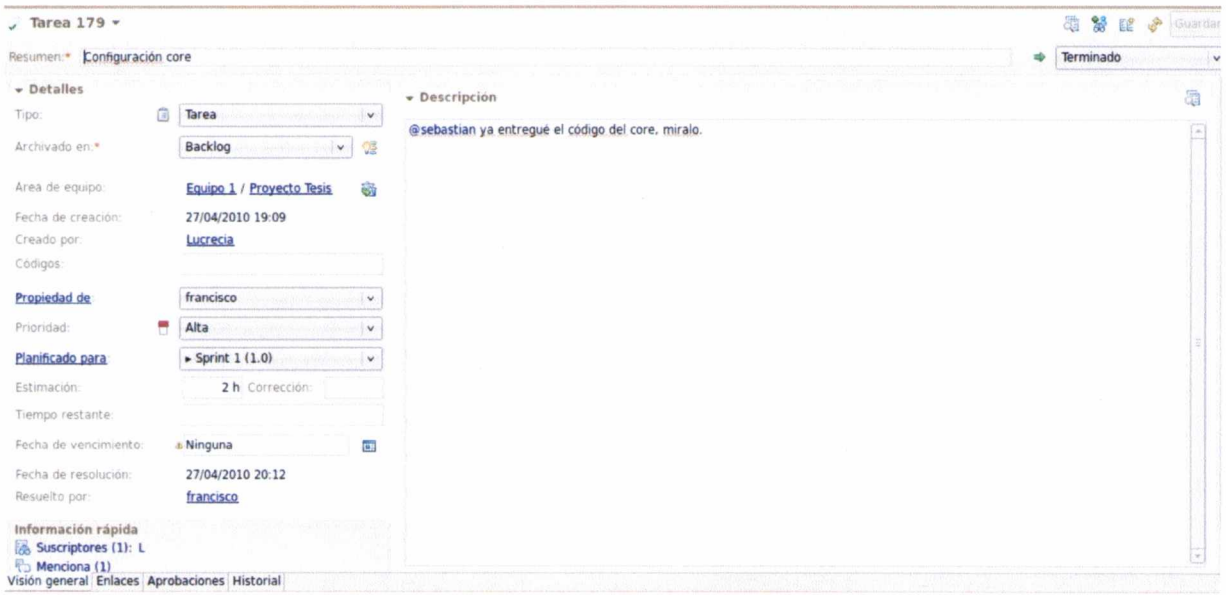

Figura 6.38. Inicio de una conversación

Y en la figura 6.39. se muestra la notificación que recibe el usuario "Sebastian".

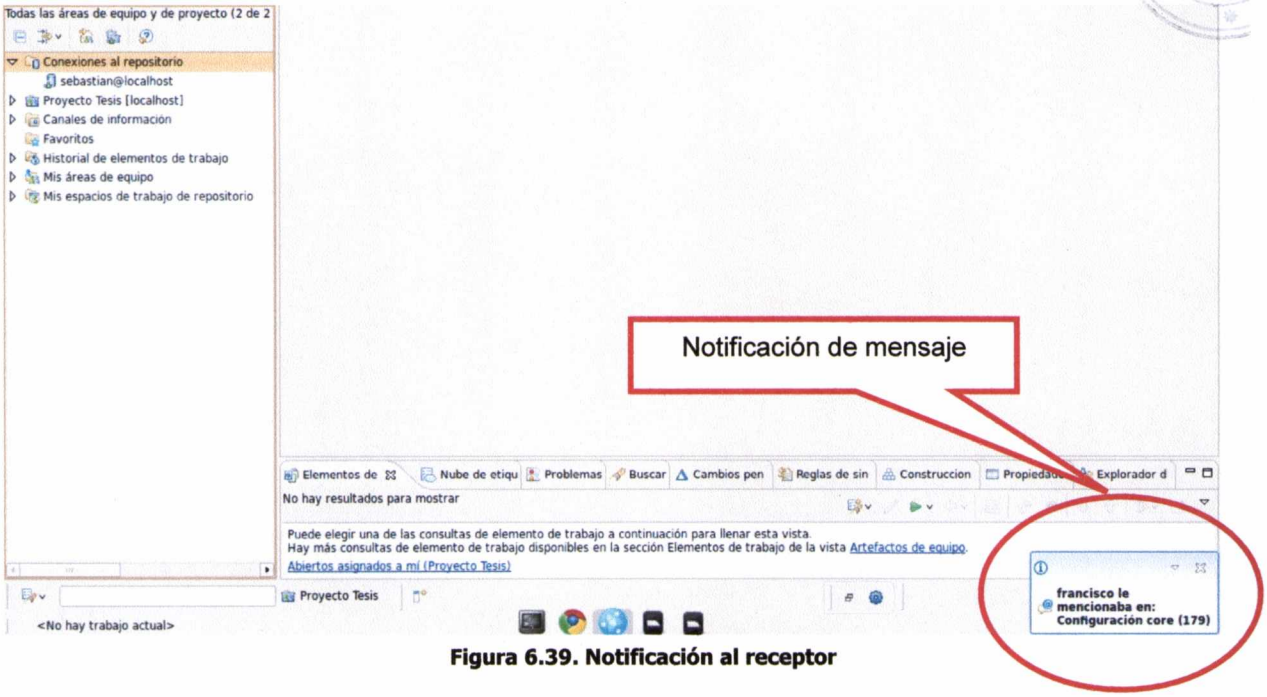

Figura 6.39. Notificación al receptor

También se pueden recibir notificaciones cuando una corriente de equipo entrega código o hace modificaciones… esto puede verse en la figura 6.40.

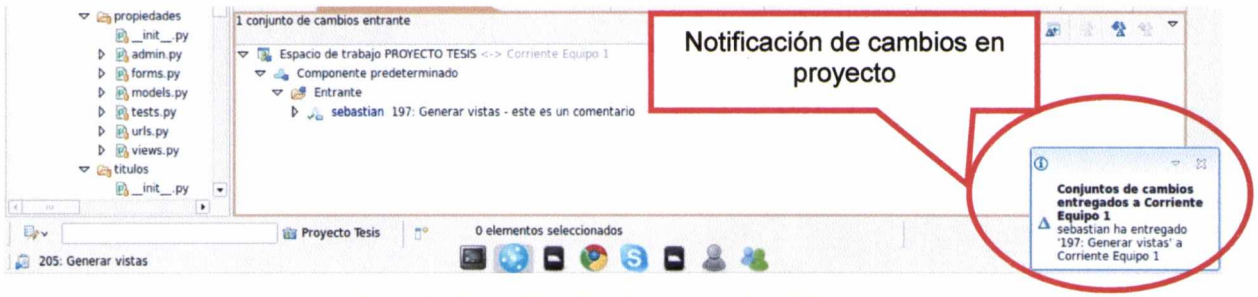

Figura 6.40. Notificación de entrega de código

Y por ultimo se muestra un ejemplo en el que un miembro (Lucrecia: ScrumMaster) se subscribe a una tarea perteneciente a un TeamMember y adjunta archivos en la tarea (figura 6.41.).

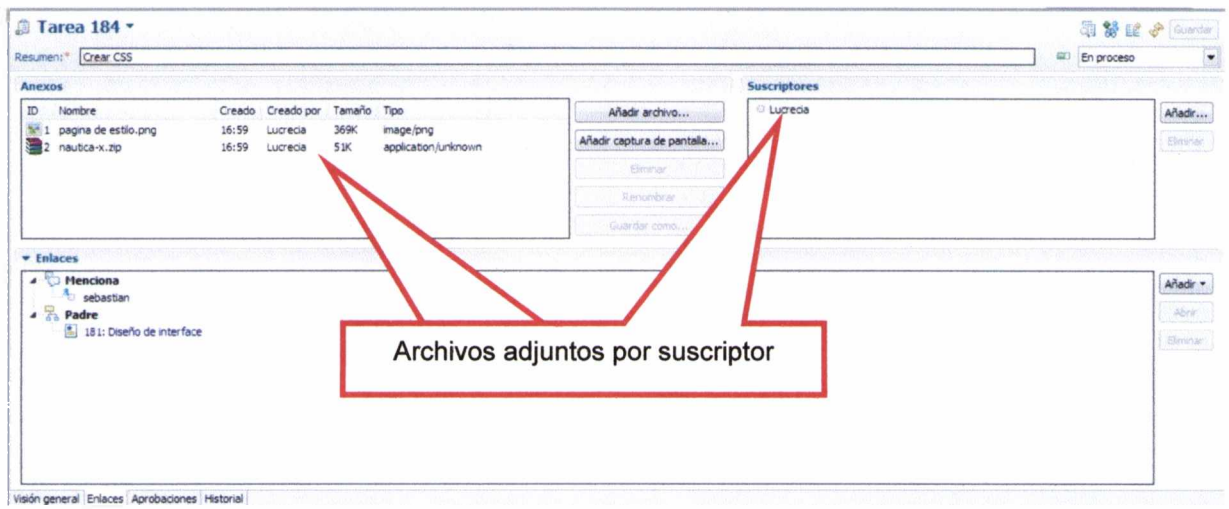

Figura 6.41. Suscripción a tarea con archivos adjuntos

Y el TeamMember puede ver la notificación en el elemento de trabajo del siguiente modo, como lo muestra la figura 6.42.

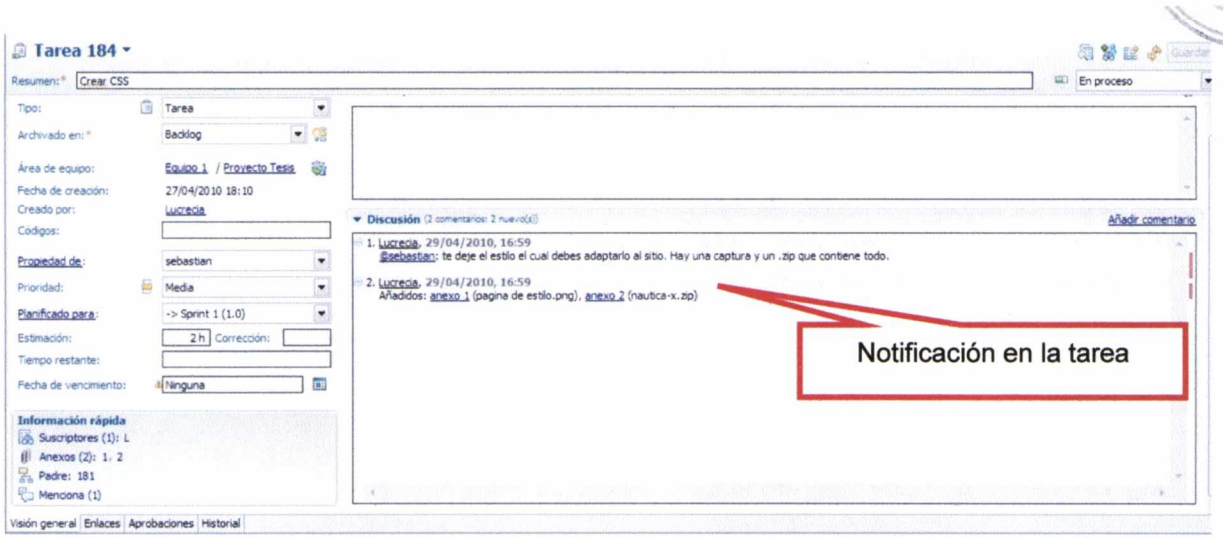

Figura 6.42. Vista de mensaje y anexos desde el receptor

También se pueden ver estas notificaciones a través del registro de eventos como lo muestra la figura 6.43.

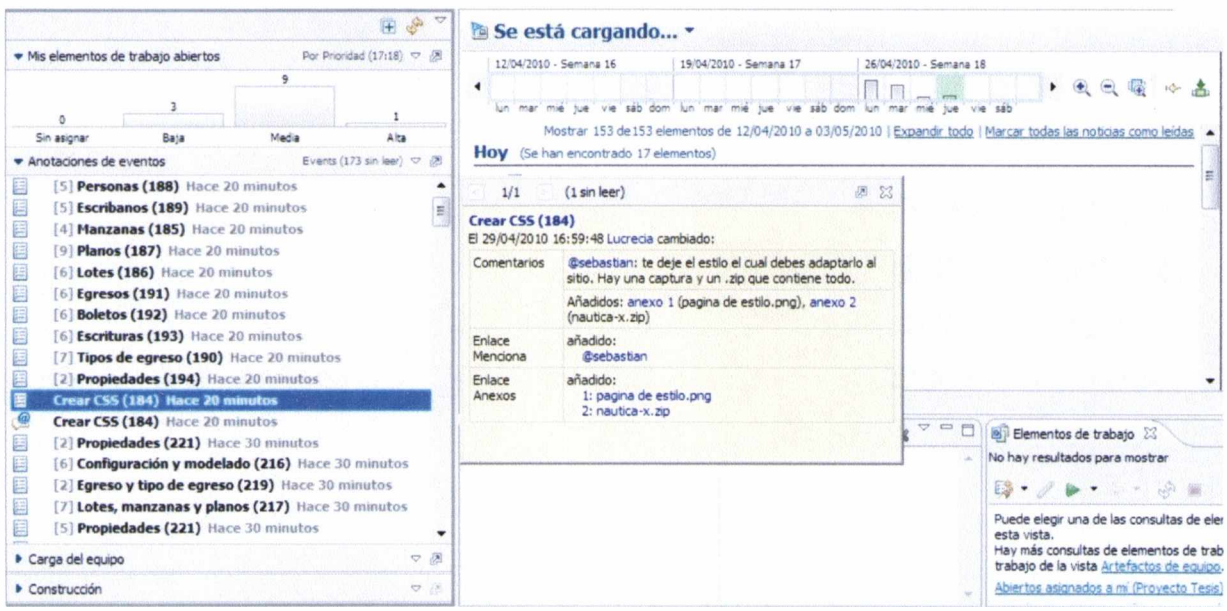

Figura 6.43. Vista de mensaje y anexos desde el receptor en el historial de eventos

Otro aspecto importante a mencionar respecto a comunicación es que el usuario puede configurar su cuenta de mail para recibir la información que desee, por ejemplo, información acerca de las tareas en las que esta suscripto.

6.5.12. Ejecución: Respecto a los informes

Si hay una característica y ventaja preponderante en RTC son los informes. La amplia variedad que otorga el soft, junto con la exactitud y la utilidad que otorgan, a nuestro entender, hacen que esta característica sea la ventaja más importante que otorga la aplicación. Para su generación RTC utiliza el motor de informes BIRT.

Existen informes de todo tipo, destinados a todos los miembros con cualquier rol.

Hay informes mas genéricos y otros mas puntuales. Son informes muy importantes para que el Owner vea el progreso de su proyecto (o del proyecto al cual representa) al igual que para los StakeHolder; sirve para que el ScrumMaster vea los progresos del equipo, los progresos y desempeños particulares, que conozca si hay obstáculos en el equipo para deshacerse de ellos, conozca si se están realizando estimaciones correctas, en caso que se manejen versionados, también saber el progreso del proyecto, etc.; los TeamMember pueden conocer

el estado de su desempeño, y compararlo con el resto del equipo para conocer su performance, entre otras ventajas.

A continuación se muestran ejemplos de los dos principales gráficos:

En la figura 6.44. puede verse un ejemplo del gráfico de avance del proyecto en el Sprint 1.

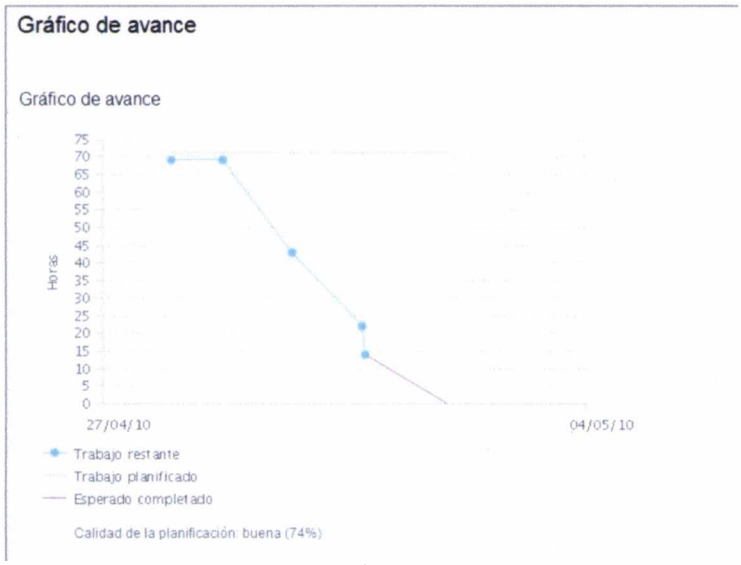

Figura 6.44. Gráfico de avance

En la figura 6.45. puede verse un ejemplo del gráfico de velocidad de equipo en el Sprint 1.

Figura 6.45. Velocidad de equipo

Ahora, en la figura 6.46. se muestra otro ejemplo de la velocidad de equipo, pero contrastando el grafico con el sprint 2 en progreso, y el grafico final con el proyecto concluido.

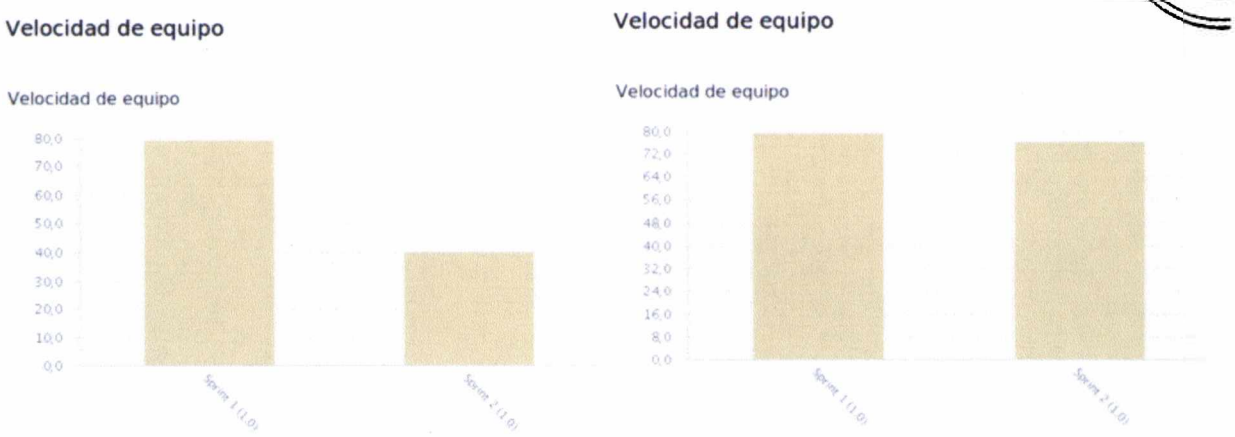

Figura 6.46. Velocidad de equipo, antes y luego de finalizar

Finalmente, en la figura 6.47. se muestra un ejemplo con similares premisas al anterior, pero con el gráfico de avance.

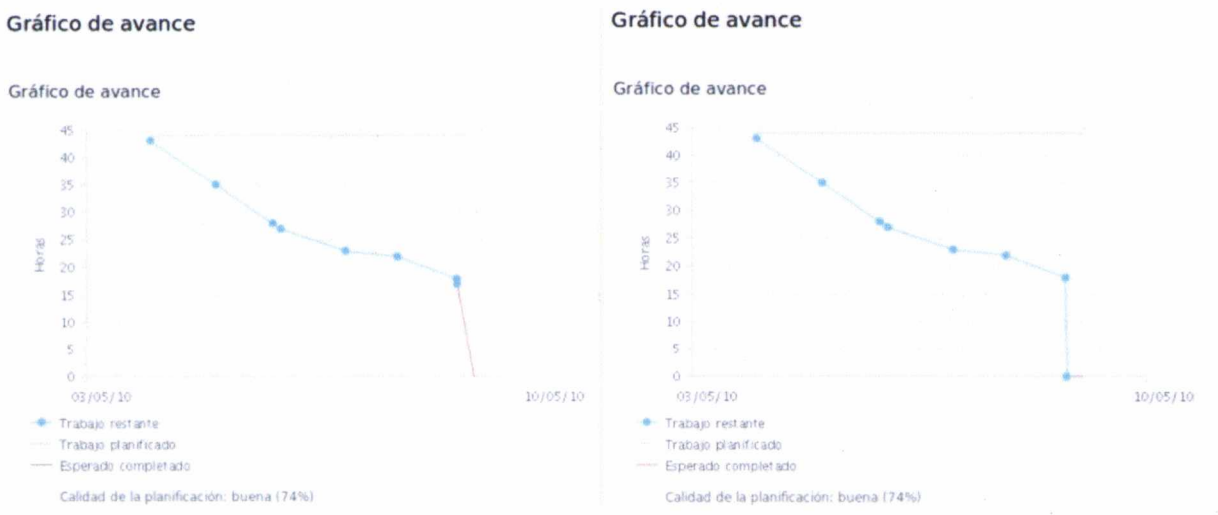

Figura 6.47. Grafico de avance, antes y luego de finalizar

6.5.13. Ejecución: Respecto al acceso Web

Otra de las Ventajas importantes de RTC, es que no depende pura y exclusivamente del IDE del cliente (Eclipse o Visual Estudio). También puede ingresarse a través de la plataforma Web.

Las funcionalidad que otorga la plataforma Web es prácticamente la misma que las aplicaciones clientes. La principal diferencia radica en que es mas accesible lo cual hace que personas que sean reacias a utilizar el cliente, como podría ser el Product Owner, accedan directamente a la plataforma Web y monitoricen desde ahí el estado de sus proyectos.

La plataforma Web ofrece DashBoards (tableros) personalizados, según el rol y las preferencias del usuario, de modo tal que cada miembro puede organizar y seleccionar la información que crea conveniente por su papel dentro del equipo.

En la figura 6.48. se puede ver un ejemplo sobre como se muestra la interfaz Web por defecto (después se puede personalizar) al Product Owner (Usuario: Joaquin).

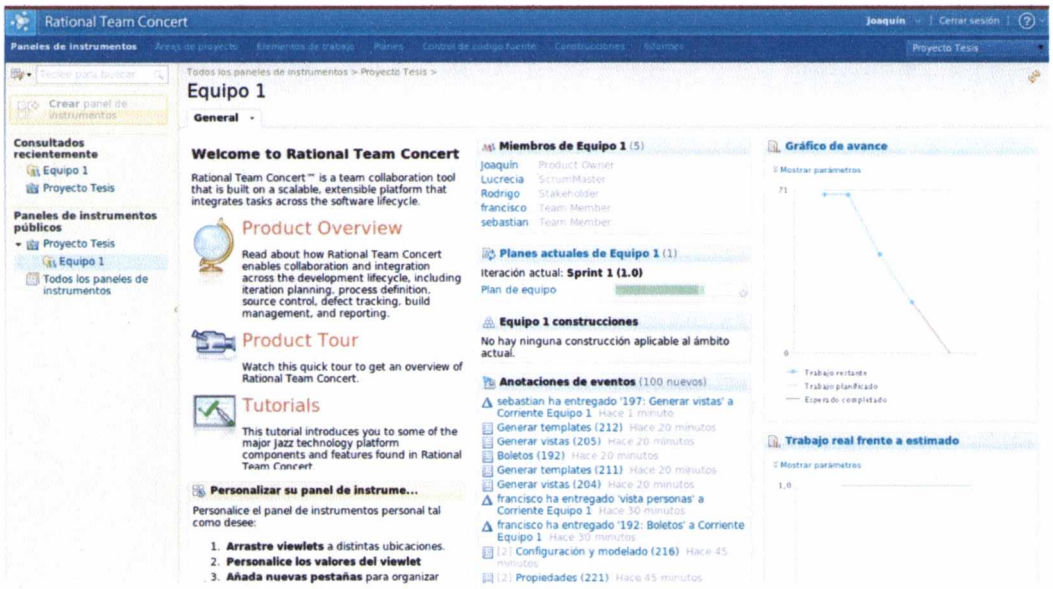

Figura 6.48. Interfaz Web del Product Owner

Y comparado con lo anterior, en el otro extremo, en la figura 6.49. cómo se muestra la interfaz al StakeHolder.

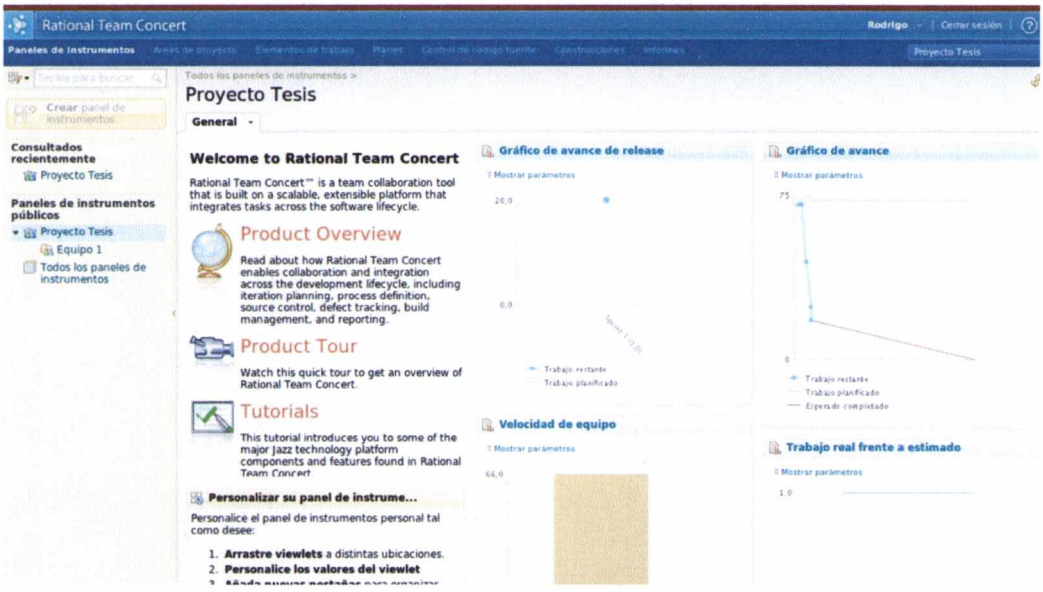

Figura 6.49. Interfaz Web del StakeHolder

También se muestra un ejemplo de un Dashboard personalizado desde la visión del ScrumMaster, donde el usuario seleccionó la información y los informes que eran de su interés. Esto puede verse en al figura 6.50.

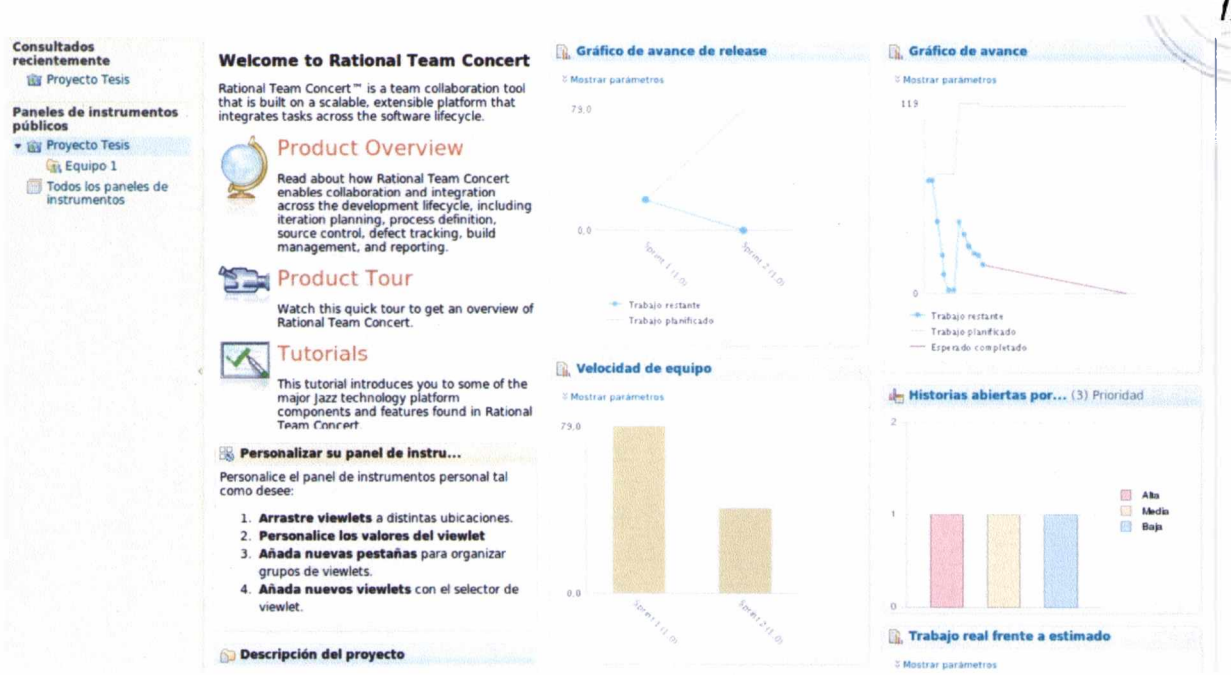

Figura 6.50. Dashboard personalizado del ScrumMaster

En la figura 6.51. se pretende mostrar que tantos las vistas como la funcionalidad que otorga el IDE es muy similar a la presentada por la interfaz Web. En el ejemplo se muestra el plan de equipo desde la visión del ScrumMaster

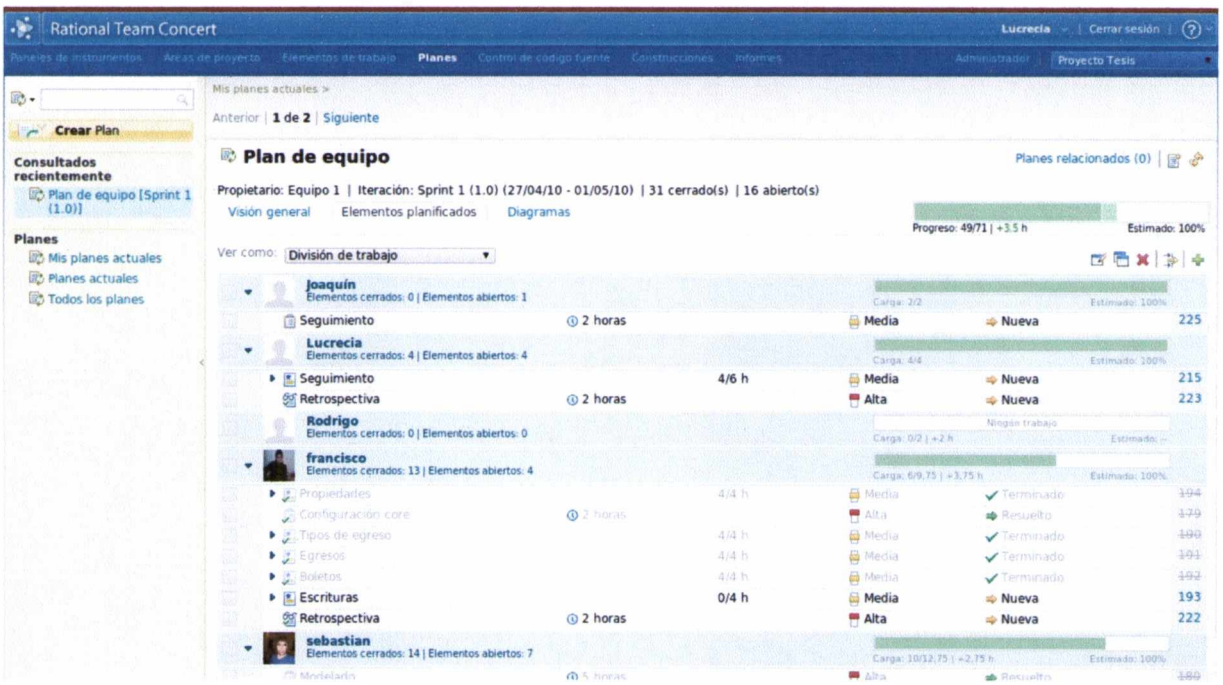

Figura 6.51. Plan de equipo desde el ScrumMaster

En la figura 6.52. se pretende mostrar un ejemplo de las restricciones que tiene el StakeHolder, desde la interfaz Web. (Usuario: Rodrigo)

Figura 6.52. Restricciones con StakeHolder

6.5.14. Proyecto finalizado

Como final del apartado de Scrum, se muestra en la figura 6.53. el resultado final del caso de prueba elegido.

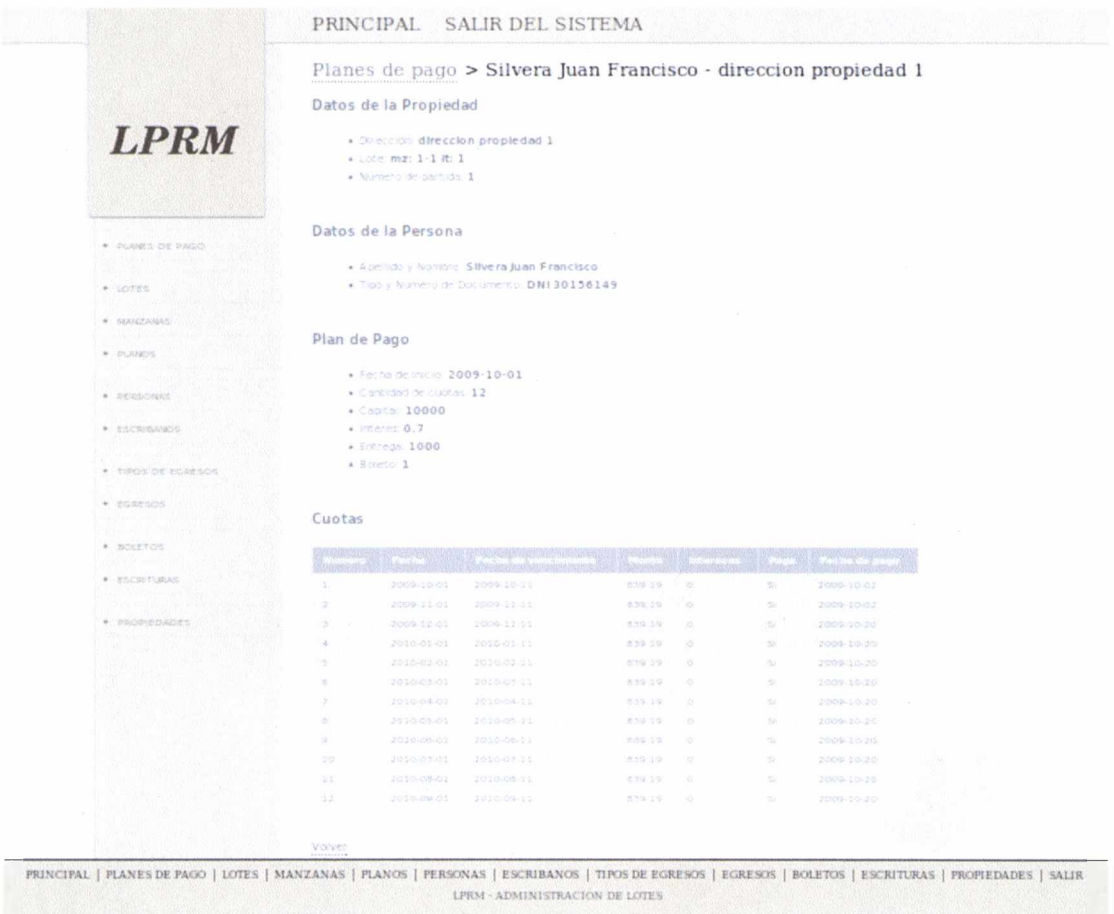

Figura 6.53. LPRM finalizado

6.6. Caso de estudio basado en OpenUp

En este apartado se explicaran las pruebas realizadas específicamente utilizando el template del proceso OpenUp.

Se realizó un caso de estudio sencillo ya que el template de éste proceso solo difiere del de Scrum en aspectos tales como la distribución de información y planes, roles de usuarios, tipos de elementos de trabajo, y otras pequeñas diferencias, que no marcan una diferencia funcional.

Se mostrarán características propias de la plantilla OpenUp y aquellos aspectos que difieren del template Scrum.

6.6.1 Sobre el área de proyecto, la plantilla OpenUp, los roles y las iteraciones

Cuando se crea el área de proyecto con el template OpenUp, la primera impresión indica que la plantilla del proceso es muy similar a la presentada en Scrum. Sin embargo, se encuentran algunas diferencias.

Respecto a los usuarios se puede mencionar que naturalmente se manejan roles propios del proceso OpenUp. Estos son "Gestor de proyecto", "Analista", "Desarrollador", "Arquitecto", "Probador" y "Parte Interesada". En nuestro caso se añadieron nuevos usuarios para poder compensar en numero la cantidad necesaria de personas para cubrir todos los roles.

Respecto a las iteraciones, se advierte que se organizan de diferente forma que en Scrum, lo cual es correcto, ya que en OpenUp se definen 4 tipos de iteraciones que funcionan de forma solapada y continúa que son: Inicio, elaboración, desarrollo y transición. Debe tenerse en cuenta que estos tipos de iteraciones son dependientes (Por ejemplo, no puede iniciarse la etapa de elaboración, hasta que no este terminada la etapa de inicio), por lo que deben configurarse muchos planes de corto plazo por cada tipo de iteración, de modo que luego entonces sí se solapen, y exploten las ventajas de OpenUp. Otra observación, no menos importante, es que no necesariamente en cada etapa intervienen todos los miembros (por su rol); Si bien es cierto que por defecto se permite a cualquier miembro intervenir en cualquier plan, cabe recordar que RTC permite configurar permisos, incluso por iteración.

Las configuraciones y los permisos que presenta la plantilla naturalmente difieren de los que presenta Scrum respecto a roles, y elementos de trabajo, pero al mismo tiempo son similares en la mayoría de los ítems.

En la figura 6.54. se muestra un ejemplo de la plantilla OpenUp.

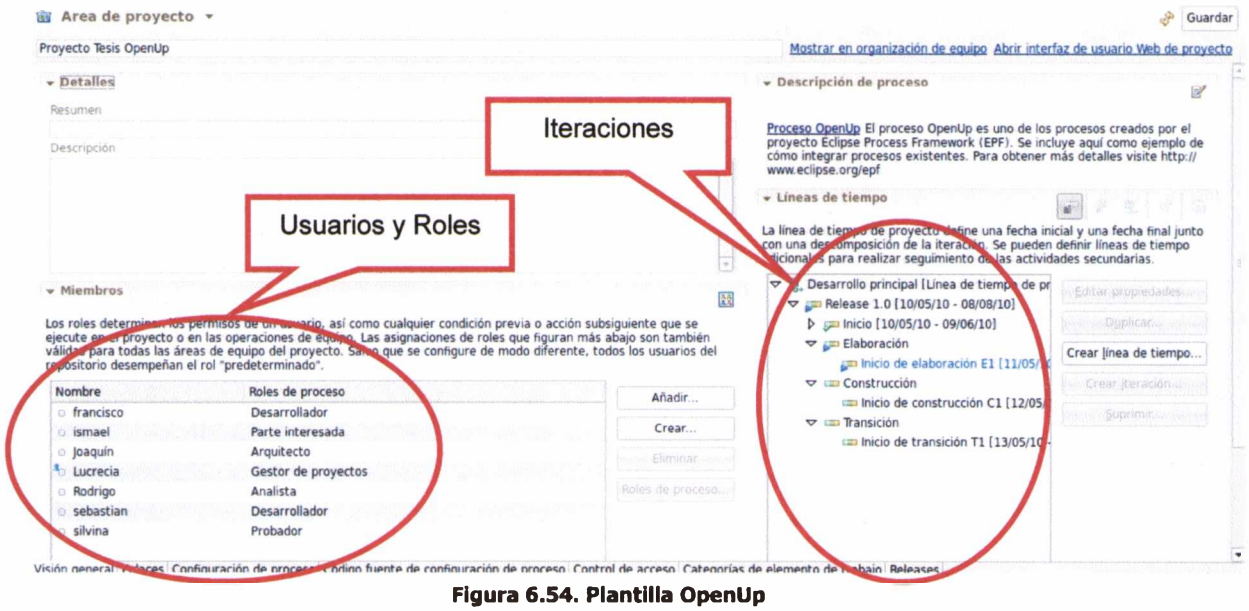

Figura 6.54. Plantilla OpenUp

6.6.2 Respecto a los elementos de trabajo

En relación a los elementos de trabajo se advirtió que son muy distintos a los que maneja Scrum.

En OpenUp se manejan Work Items tales como "Tarea", "Caso de uso", "Defecto", "Mejora" y "Riesgo". Si bien hay muchas propiedades que se mantienen, como son el propietario, la fecha de creación, etc., hay campos que difieren de los elementos de trabajo de la plantilla Scrum y otros que no están.

En el caso del "Caso de uso" se pueden definir campos como "condición posterior" y "precondición".

En el caso del "Riesgo" se definen campos como "probabilidad" e "impacto".

En el caso de la "Mejora", la "Tarea" y el "Defecto", se advirtió que tienen las mismas propiedades. Se manejan campos como "Gravedad" y "Encontrado en". En este sentido, es al menos curioso que la "Mejora" y la "Tarea" tengan estos campos ya que no parecen ser coherentes con lo que definen; parecen campos propios de un "Defecto".

En ninguno de los casos existe el campo "prioridad".

En la figura 6.55. se muestra un ejemplo de un "Riesgo".

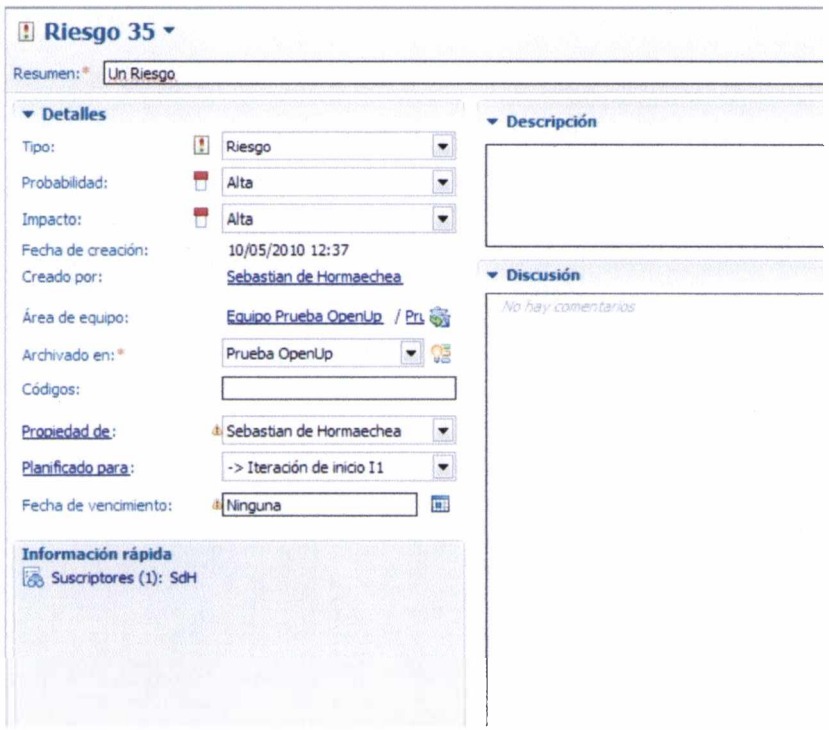

Figura 6.55. Riesgo

Finalmente en la figura 6.56. se muestra, a modo de ejemplo, que la estructura de una "Tarea" y un "Defecto" son idénticas.

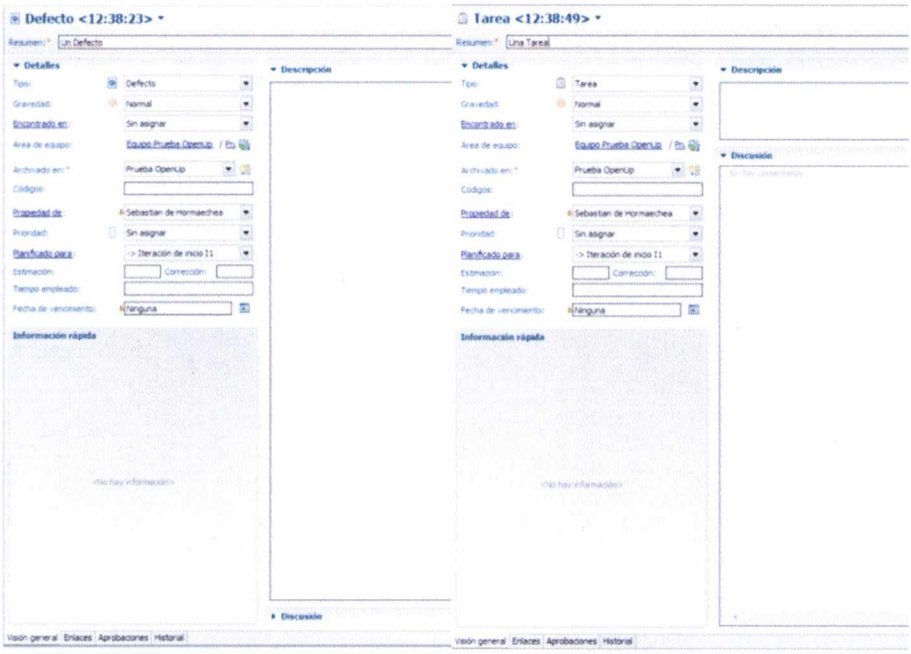

Figura 6.56. Tarea y defecto idénticos

6.6.3 Respecto a los planes

Pese a que cambia la estructura de los planes por los tipos de iteración que OpenUp sugiere, los planes se visualizan, se ejecutan y proveen la misma automatización que los planes desarrollados en Scrum.

Cabe aclarar que la plantilla OpenUp por defecto no respeta la definición del proceso; No se respeta la definición misma que RTC provee del proceso, donde se sugiere que no todos los miembros participen en todas las etapas (por ejemplo, los "Desarrolladores", no tienen tareas a desarrollar en la iteración de "Inicio", según esta definición).

En la figura 6.57. se muestra una ejemplo de la estructura de los planes.

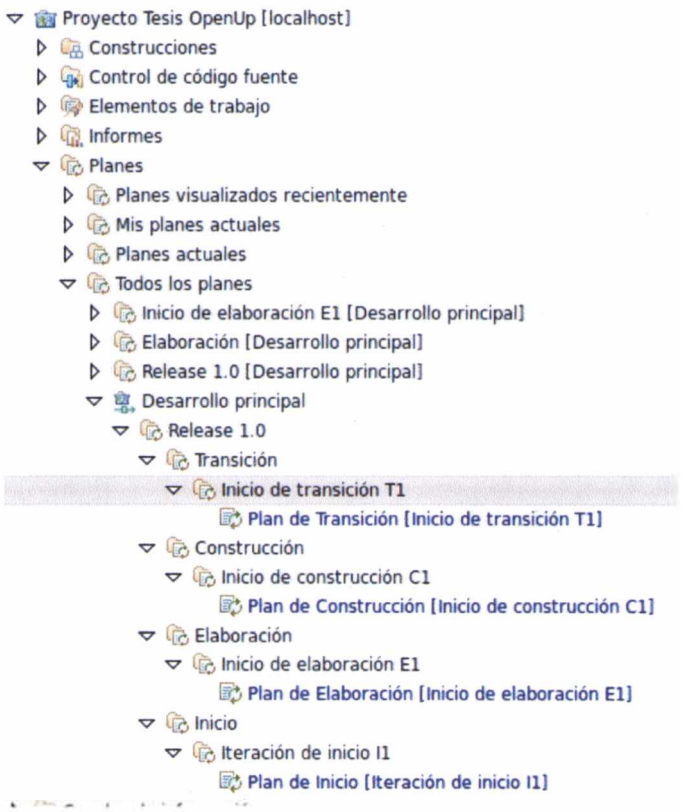

Figura 6.57. Estructura de los planes

A continuación se mostrarán ejemplos de los 4 tipos de planes.

En la figura 6.58. se muestra el plan de "Inicio".

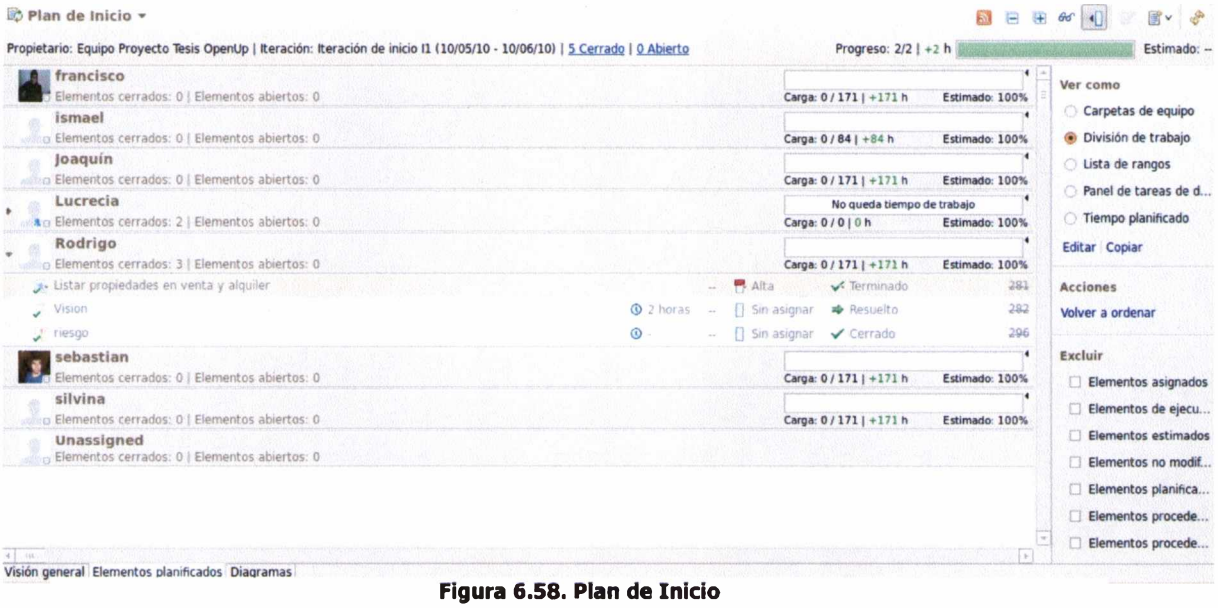

Figura 6.58. Plan de Inicio

En la figura 6.59. se muestra el plan de "Elaboración".

Figura 6.59. Plan de Elaboración

En la figura 6.60. se muestra el plan de "Construcción".

Figura 6.60. Plan de Construcción

En la figura 6.61. se muestra el plan de "Transición".

Figura 6.61. Plan de Transición

6.6.4 Otras características

Como ya se mencionó RTC provee todas las funcionalidades ya explayadas, las cuales son indistintas a la plantilla de proceso que se utilice.

Se dispone de una comunicación íntegra (suscripciones, mensajes instantáneos, mails, etc.), control de código fuente, alta disponibilidad de reporting (incluyendo informes diferentes a los proporcionados por Scrum), consultas predefinidas y personalizadas, relaciones de tareas, configuración de horas de trabajo, carga estimada por área y ausencias planificadas, posibilidad de acceso por la interfaz Web, herramientas y funcionalidades útiles que provee RTC, alta flexibilidad en general, etc.

En la figura 6.62. se muestra un ejemplo del gráfico de avance en OpenUp.

Figura 6.62. Grafico de avance en OpenUp

6.7. Otras pruebas y consideraciones genéricas

En este apartado se muestran otros aspectos con los que nos encontramos en medio de la ejecución.

Nos encontramos con algunos errores donde resulta muy difícil descubrir el origen del error ya que la excepción no esta tratada debidamente para que el usuario pueda entenderla. Se muestran los errores por defecto de Base de Datos o de Java.

Con estos tipos de errores técnicos nos encontramos frecuentemente. Errores inesperados que, luego de repetir los pasos que originaron el error, no volvían a surgir. También hubo casos en los que se optó por reiniciar la aplicación cliente, y tampoco volvían a aparecer esos errores.

Como ejemplo, se muestra el caso de un error interno de RTC que surgió al intentar eliminar un espacio de trabajo. (figura 6.63)

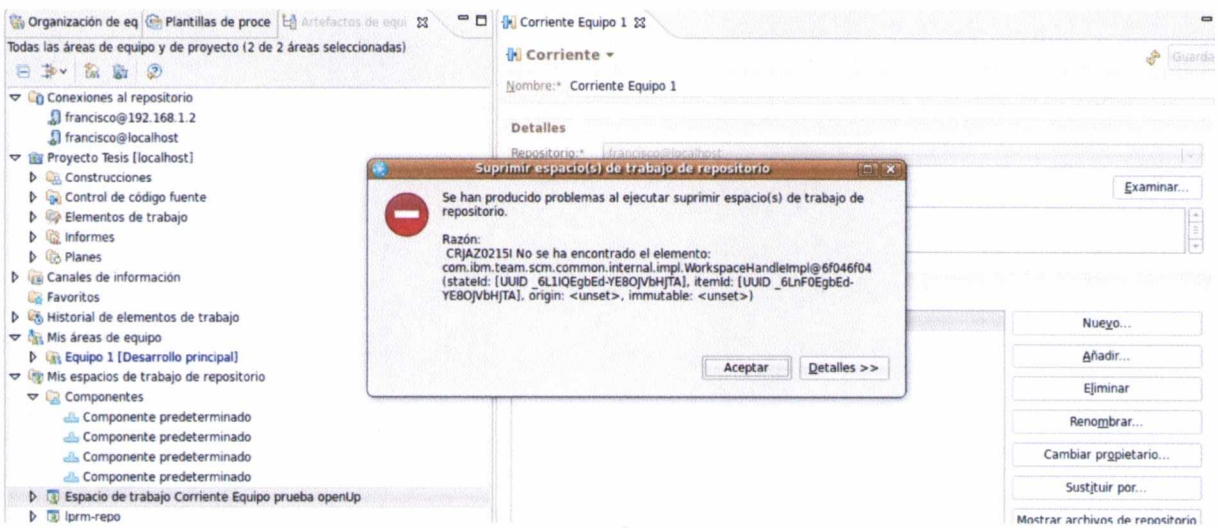

Figura 6.63. Error al intentar eliminar espacio de trabajo

Otro aspecto para remarcar es que pese a conocer la IDE de Eclipse y pese a entender el funcionamiento de Scrum y OpenUp, la interfaz junto con la nomenclatura utilizada, no nos

resultaron muy intuitivas para entender el software. Nos referimos a que RTC utiliza una terminología y una organización de la información que excede los conocimientos sobre los procesos o sobre el IDE Eclipse. Términos como "componente", "corriente", "espacio de trabajo de repositorio", solapas como "Artefactos de equipo", "Central de equipo", "Organización de equipo", "Mi Equipo", entre varios otros resultaron muy confusos en un principio. Se tuvo que recurrir frecuentemente a la ayuda, en la cual, la información no es fácil de encontrar, o a referencias Web para apoyarnos.

De todas maneras, una vez superados los inconvenientes y entendiendo la nomenclatura de RTC, y sus funcionalidades, se puede destacar que la aplicación ofrece muchos atajos a los usuarios para realizar las mismas acciones, lo cual es una ventaja importante del software.

RTC ofrece también la posibilidad de realizar Builds. En nuestro caso se intento configurar el software para realizarlas con el motor de construcción Ant. Pero en el paquete RTC instalado Ant solo soporta proyectos JAVA, y nuestro proyecto fue realizado en Python. Respecto a esto se quiere remarcar que el software debería ser totalmente independiente del código en el que se realicen los proyectos y deberían estar todas las opciones incluidas en el paquete de instalación de RTC.

Por otra parte, se puede configurar un servidor de respaldo, que tiene una copia idéntica del primero. En caso que el servidor principal falle se utilizará un servidor de respaldo.

RTC también se puede configurar para que se soliciten aceptaciones a miembros de equipo para finalizar un Work Item; también se puede configurar para que se soliciten construcciones de código fuente.

En la configuración del proyecto, RTC permite configurar que no se finalicen Works Items que tengan elementos "hijos" sin finalizar, lo cual implícitamente sugiere que de éste modo se pueden simular dependencias reales, y por lo tanto, "caminos críticos".

RTC provee un sistema de control de versiones propietario, Jazz Source Control; sin embargo soporta sistemas conocidos como SVN.

La flexibilidad y la gran representatividad que otorga RTC, como ya se dijo, como contrapartida genera una deficiencia en la funcionalidad de aquello que representa. RTC permite configurar incluso datos incoherentes, sin proveer ningún feedback o advertencia al usuario. También se advierte la ausencia de eventos automáticos. Se pueden nombrar como ejemplos:

- La relaciones de cualquier elemento de trabajo con cualquier otro (sea coherente esa relación o no)
- En los planes de las iteraciones se pueden definir "Historias" que tienen "Tareas". Aun cuando se finalicen y terminan todas las tareas, se debe finalizar manualmente la "Historia", siendo que éste evento podría dispararse automáticamente. Del mismo modo, por default, se permite finalizar la historia sin finalizar las tareas hijas de la historia (de todos modos se puede configurar para no permitir esta acción).
- Otro caso, como se muestra en la figura 6.64. es la creación de una jerarquía de iteraciones (Sprint) donde la iteración "hija" puede no respetar la fecha de inicio o de finalización de la iteración "Padre".

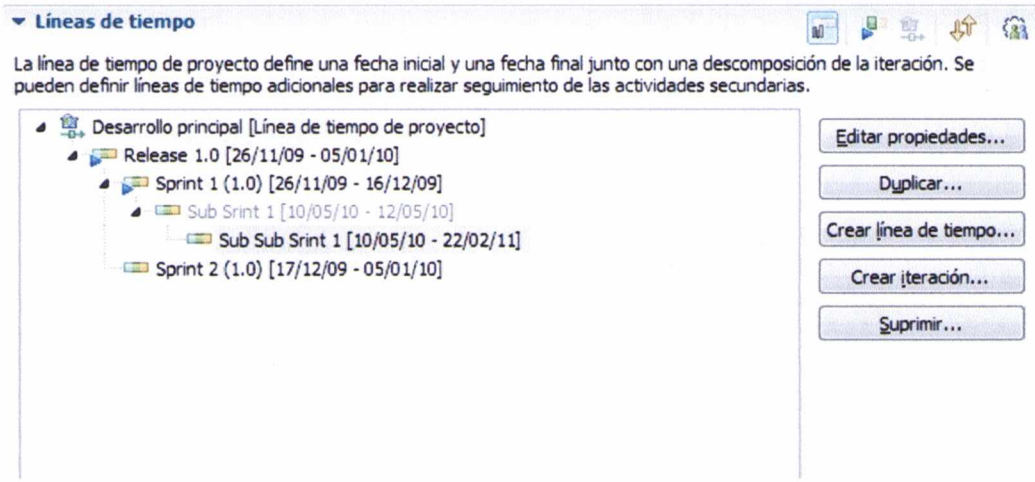

Figura 6.64. Creación de iteraciones y sub-iteraciones incoherentes

Como ultima observación sobre RTC, se quiere remarcar una ventaja como lo son los diagramas de flujo. Los diagramas de flujo ayudan a visualizar la relación de los espacios de trabajo y las corrientes en el Jazz. Se utilizan estos diagramas para ver los flujos de entrada y salida entre los espacios de trabajo y las corrientes. También se pueden realizar determinadas acciones que afectan al repositorio, por ejemplo, crear o suprimir corrientes o espacios de trabajo, lo cual implica otro atajo para los usuarios. (Figura 6.65.)

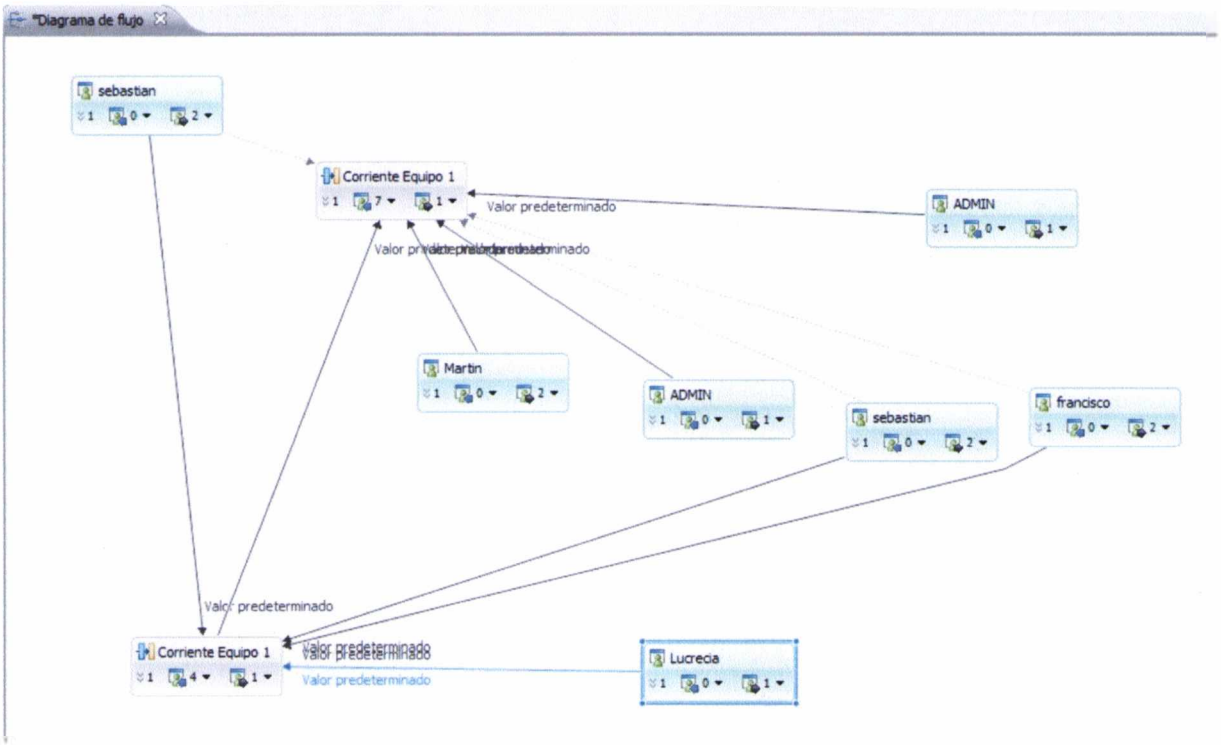

Figura 6.65. Diagrama de flujos

Capítulo 7

Conclusiones

En la actualidad, las organizaciones de desarrollo de software están sometidas a crecientes presiones para suministrar rápidamente software de alta calidad coherente con los objetivos, que se modifican y mutan constantemente. No resulta fácil realizar esto por varios factores. Se exige a los equipos crear más aplicaciones con menos recursos; muchos equipos están dispersos geográficamente, lo que complica la colaboración, y genera problemas para aplicar procesos coherentes en grupos dispares.

El desarrollo de software requiere de un enfoque orientado al equipo en lugar de esfuerzos individuales, donde exista una estrecha colaboración entre los miembros (aun si están dispersos geográficamente) para lograr una eficiencia absoluta, ya que el ciclo de desarrollo es cada vez más comprimido, porque las empresas se esfuerzan por conseguir la mayor ventaja competitiva en el lanzamiento de sus aplicaciones.

Rational Team Concert se presenta como una herramienta innovadora que permite el manejo de ciclo de vida de aplicaciones ágiles (mediante la utilización de Scrum, OpenUp, etc.), que dispone de un ambiente colaborativo entre los miembros de un equipo (que pueden estar en diferentes lugares físicos), que proporciona un ámbito de desarrollo transparente a los usuarios, etc.

Habiendo adquirido los conocimientos para poder realizar un análisis crítico de la herramienta, concluimos que efectivamente se trata de una aplicación innovadora, donde se integran innumerables funcionalidades, y herramientas.

Se advirtieron diversas ventajas que hacen a la aplicación una fuerte herramienta para la administración de proyectos:

- Buena performance de acceso y comunicación aun con conexiones como, por ejemplo, Wi-fi. No consume muchos recursos de red, ni necesita consumir mucha memoria
- Una de las ventajas más importantes que provee el software, es la gran flexibilidad que provee en, prácticamente, todo aquello que implementa. Flexibilidad en las iteraciones y en los planes, donde se pueden configurar Work Items "padres" e "hijos", donde se puede configurar reglas para los usuarios (como no dejar entregar código fuente que falle en compilación o sin asociarlo a una tarea), etc.; flexibilidad en los usuarios, donde se pueden configurar roles, licencias, horas de trabajo, carga por área, comunicación por mail; flexibilidad en la administración de proyectos; flexibilidad en la alta y completa administración de permisos que provee; flexibilidad en la comunicación entre usuarios y con el usuario ya que cada persona decide que información desea recibir y el modo en recibirla (por mail por ejemplo); flexibilidad en las vistas de diferentes secciones, sea de los planes, de los equipos, de las áreas de proyecto, etc.; flexibilidad en las consultas, ya que un usuario puede crear sus propias consultas y guardarlas; flexibilidad en los elementos de trabajo y sus transiciones, ya que se pueden crear tanto elementos como transiciones nuevas, o personalizar los Works Items ya existentes; incluso flexibilidad en las plantillas de proceso, ya que cada usuario puede crear y personalizar su propio proceso de desarrollo; flexibilidad y personalización de los Dashboards en la interfaz Web. La flexibilidad del software aporta una amplia representación y configuración, y favorece la usabilidad de la aplicación.
- Quizás la ventaja más interesante y más importante sea el excelente motor de informes de RTC. La aplicación aporta una amplia, correcta y fructífera variedad de informes que proporcionan muy importante información para todas las personas involucradas en un proyecto, que exceden ampliamente, por ejemplo, a los informes sugeridos por el proceso Scrum (Gráfico de avance, Gráfico de avance de release, Punto de historia por iteraciones, Velocidad de equipo).
- Otra ventaja a remarcar, es la comunicación que proporciona el software. Los usuarios tiene variadas formas de comunicarse entre si, y de forma instantánea. Las suscripciones a elementos de trabajo, a usuarios, a un plan, a eventos generados por un usuario, proporcionan una importante herramienta de información que tienen los miembros para estar al tanto de los eventos que sean de su interés. Sumado a la comunicación instantánea entre usuarios y al alto registro de eventos de RTC, la aplicación proporciona el ambiente para una estrecha colaboración.
- RTC tiene una alta capacidad de registro. Se registran historiales de planes, tareas, usuarios, áreas de equipo, proyectos, eventos de usuarios, eventos sobre código fuente, etc. Estos registros pueden visualizarse de forma amigable como por ejemplo, los diagramas de flujo para conocer los espacios de trabajos y las corrientes de equipo.
- La posibilidad de un usuario en intervenir en varios equipos y/o proyectos, la posibilidad de configurar sus horas de trabajo (incluso ausencias planificadas) y de estimar la carga de trabajo por área, la posibilidad de asociar áreas de equipo a categorías de elementos, la posibilidad de administras proyectos y procesos en general, la posibilidad de priorizar elementos de trabajo, entre otros, hacen a RTC una herramienta que se acopla muy bien a situaciones reales y cotidianas. Además la utilización y asociación de toda esa información para, por ejemplo, registrar el progreso de un miembro, o de un área de proyecto o equipo, o para sugerir a un miembro una planificación de tareas futuras a realizar, hacen a RTC una herramienta inteligente y muy útil en este aspecto, que puede ayudar a una mejor organización de los miembros. Queda evidenciada una alta asociatividad de diferentes objetos, junto a una frondosa BD.

- RTC también provee herramientas y funcionalidades muy útiles como son las consultas sobre elementos de trabajo y los filtros, las instantáneas, el visor de conflicto de código para fusionar archivos, etc., que facilitan el trabajo de los usuarios.
- RTC provee muchos atajos para realizar distintas acciones, o acceder a distintas secciones.
- RTC también tiene la ventaja de poder ser administrado tanto desde el IDE como desde la interfaz Web.
- RTC aprovecha todas las ventajas del entorno de administración y el entorno de desarrollo que brinda Eclipse. Con una buena capacitación acerca de los conceptos que maneja RTC, se logra un manejo intuitivo de la interfaz y sus elementos.

Sin embargo, con la experiencia de haber utilizado la aplicación y a pesar de las excelentes ventajas que otorga el software, también se advirtieron, al menos observaciones no menos importantes (no necesariamente desventajas):

- Pese a poder tener un servidor de respaldo, se trata de una aplicación que utiliza un servidor centralizado, con los problemas de disponibilidad y de comunicación que esto podría implicar
- Respecto a lo funcional, en algunas situaciones, RTC no respondió como se esperaba intuitivamente, lo cual puede implicar un serio problema de fiabilidad. Creemos que RTC no provee de un feedback completo: Hay errores que no están tratados, mostrando códigos originales de JAVA o de BD; Hay errores que cuando se piden los detalles, muestra el mismo error. Modificando la fecha del sprint, luego de cargados todos los datos, se pierde el plan completo sin realizar siquiera ninguna advertencia acerca de tan gran problema. Tampoco se advierte cuando se configuran Sprint "padres" e "hijos" con inconsistencia en las fechas. Por otra parte, no automatiza algunas cosas esperadas, como la finalización automática de una historia cuando se terminan todas sus tareas, entre otros.
- Como contrapartida a la gran flexibilidad y gran poder de representación que tiene RTC, se puede mencionar la poca funcionalidad de algunos elementos que se pretenden representar (por ejemplo, el elemento de trabajo "Retrospectiva", las dependencias y los bloqueos entre elementos de trabajo, etc.).
- Si bien se comprende que la aplicación esta realizada para soportar grupos medianos o grandes de desarrollo, la confusión de roles entre el área de equipo y el área de proyecto existe. No creemos correcta esta administración de roles ya que implícitamente se están creando mas roles de los que sugiere, por ejemplo, Scrum. Quizás la utilización de otra nomenclatura para los roles en el área de proyecto, sea la forma correcta de disipar la confusión.
- Ayuda difícil de encontrar: Si bien, la ayuda parece ser muy completa, hubo casos en los cuales se dificultó mucho encontrar algunos temas.

Respecto a las características que debe tener un buen software como Corrección, Fiabilidad, Eficiencia, Flexibilidad, Portabilidad, Interoperabilidad, Integridad y Facilidad de uso, se puede concluir que:

- Los fundamentos que presenta IBM para lanzar el software son colaboración, reportes y automatización. Respecto a la *corrección* del software, se cumple muy bien con los dos primeros, pero, si bien no se puede afirmar que es un problema de correctitud, si puede afirmarse que falta refinar un poco la automatización.
- La *fiabilidad* es, al menos, muy discutible. Según nuestra percepción y nuestra experiencia, el software no resulta completamente fiable, ya que no responde como uno intuitivamente espera.

- Se comprobó que RTC es *eficiente*, ya que se probó con distintos tipos de conexiones, y ante eventuales cambios, el software respondió con rapidez.
- Quedó evidenciado que RTC es altamente *flexible* por las razones anteriormente mencionadas... Es su mayor ventaja en cuanto a éstas características genéricas
- La *interoperabilidad* y la *portabilidad* son también características naturales de RTC, que aprovecha de la IDE de Eclipse, que, como se sabe, también las posee.
- Según la experiencia adquirida, y por las pruebas que se realizaron se puede mencionar que la *integridad* es una característica de RTC, ya que incluso en el peor caso que fue cuando se desarmó el plan, los elementos de trabajo no se perdieron. Además, como ya se mencionó, RTC almacena registros e historiales de prácticamente todo lo que maneja. No se advirtieron falencias en este sentido
- *Facilidad de uso:* Según nuestra experiencia, RTC no es de fácil autoaprendizaje, aún conociendo conceptos asociados a las metodologías Scrum y OpenUp, ni conociendo el IDE Eclipse. Es un punto discutible ya que luego de entender más profundamente los conceptos que maneja RTC, el desarrollo del caso de prueba resultó mucho más intuitivo. Por lo que concluimos que luego de una capacitación adecuada, su uso es intuitivo. Además proporciona la interfase Web, que creemos más amigable y apta para usuarios que no tengan experiencia en el IDE Eclipse.

Como conclusión y asociación a capítulos anteriores, se puede mencionar que:

- Si bien es entendible en algunos aspectos, RTC en su template Scrum por defecto no respeta la definición formal del proceso. En el caso del template OpenUp por defecto, sucede lo mismo, pero se cree mucho más discutible ya que se trata de un proceso simple y extensible.
- Es prácticamente imposible realizar una comparación directa entre las aplicaciones expuestas en el capitulo 4 y RTC. La mayoría de los productos son sólo sistemas de control de versiones (algunos con otras características), mientras que RTC, además, es una aplicación orientada a la planificación, la colaboración, la automatización, los motores de informes, la flexibilidad, los motores de construcción de código, etc. RTC provee un sistema de control de versiones propietario, Jazz Source Control, que difiere de otros conocidos como SVN, en que puede asociarse a los WorkItems con el conjunto de cambios que se hacen luego de un COMMIT (Entre otras diferencias). El software de similares características es el Team Foundation Server de Microsoft. Según bibliografía consultada, la principal diferencia con el Team Foundation Server de Microsoft, es que RTC se adapta y soporta productos ya existentes, mientras que TFS fue creado para soportar software nuevos de Microsoft. En una visión rápida que se hizo sobre TFS, se advirtió que representa mucho mejor los Backlog y las pizarras que RTC.
- Respecto a las características y ventajas que IBM dice que su software tiene, sólo se ponen en cierta duda aspectos tales como "Facilidad de uso" y la "fiabilidad" por cuestiones mencionadas anteriormente.

Detrás de todas las platillas de proceso que maneja RTC, hay una gran y única plantilla de proceso, muy flexible y configurable, que soporta dichos procesos. Además esta visión se refuerza, por el hecho de que los usuarios pueden generar sus propios procesos, prácticamente como deseen.

Se concluye que RTC es una aplicación apta para un grupo confiable y disciplinado; caso contrario no tiene sentido utilizar sus templates, ya que no se explotarán todos sus recursos, y conviene seguramente utilizar cualquier otra herramienta.

No parece tratarse de una herramienta para grupos muy chicos, ya que tampoco se aprovecharán todos los recursos que provee la herramienta, y nuevamente nos encontramos con el interrogante acerca de que si no conviene utilizar otra herramienta.

Como primer impulso, y por los problemas de fiabilidad que se encontraron, tampoco se cree que sea una herramienta apta para grupos muy grandes de personas, excepto que se trate de un grupo muy organizado y muy educado en la utilización de la herramienta, donde exista una capacitación previa, una administración de permisos cuidadosa y "reglas estrictas" para evitar cometer errores, y aprovechar entonces sí al máximo todas las ventajas que provee RTC.

En nuestra opinión, es un software óptimo para grupos medianos organizados, educados y disciplinados.

RTC es una aplicación excelente y completa, que necesita refinar sólo algunas características.

7.1. Entrevista

A modo ampliatorio, se realizó una entrevista a personal del laboratorio de investigación LIFIA (Vanesa Mola, Darío Silva Morán y Diego Cano), que utiliza el proceso ágil Scrum.

El objetivo de la reunión fue apuntar a que, personas con experiencia real en Scrum, nos describan cómo utilizaban y se desenvolvían en el proceso, y en que herramientas se apoyaban.

También el objetivo fue presentar la herramienta Rational Team Concert, contando y describiendo sus características, y mostrando algunos ejemplos (Especialmente del template Scrum), para conocer si lo utilizarían, advertir nuevas falencias y ventajas, con el fin de refinar y obtener nuevas conclusiones.

Los entrevistados advirtieron diversos puntos positivos como son:

- La integración del entorno de desarrollo y la administración del proyecto y del proceso
- La variedad y cantidad de informes y la posibilidad de generar nuevos informes
- La flexibilidad (se pueden agregar tareas, personalización de proceso, etc.)
- Asociación de tarea con el código fuente
- Aprobaciones
- Si bien se advirtió que se prefería un sistema de control de versiones conocido (y no propietario como es Jazz Source Control), los entrevistados se notaron satisfechos cuando se les comento que RTC soportaba SVN.
- ETC.

También advirtieron falencias o puntos negativos:

- Permite configuraciones inconsistentes (Inconsistencia de sprint "hijos" y "padres", relación incoherente de Work Items)
- No esta bien representada la pizarra en el plan. No se representa el Backlog en forma clara
- Respecto a los Work Items:
  - Faltan campos en la definición predefinida en algunos Work Items (por ejemplo en la Historia, faltan campos como "Para quién?", "Por qué la quiere?", etc.)
  - No tienen funcionalidad
  - No se puede crear un riesgo (por default)
- Si se pretende desarrollar Scrum, no se cree apto para un grupo distribuido geográficamente ya que no se pueden representar los planning, ni las retrospectivas dentro de un mismo equipo. Se necesitarían herramientas que permitan simular estas importantes reuniones propias de Scrum.

Como ratificación a este ultimo punto, retomamos la idea de que se necesita una buena educación en los usuarios para un buen uso de la aplicación (Los entrevistados también apoyaron este punto). ¿Podría lograrse esto en un grupo y entorno distribuido geográficamente, donde los involucrados probablemente se desconozcan?

Como nuevo aporte, se pudo comprobar luego de la reunión, que la creación de elementos de trabajo no es intuitiva, ni accesible, ni simple.

Conociendo aún más los conceptos relacionados al proceso Scrum, se adhiere en los puntos expuestos por los entrevistados.

Se destaca que se trató de una reunión muy fructífera que nos aportó valiosa información acerca del proceso Scrum.

Capítulo 8

Trabajos futuros

Pese a que se también se abordaron otros aspectos (en forma más genérica), el desarrollo de esta tesis se focalizó en el comportamiento del software Rational Team Concert respecto a los procesos ágiles OpenUp y Scrum.

No obstante, el software RTC presenta otras diversas propiedades que también merecen su atención y la investigación respectiva, para obtener, como ampliación de este informe, un análisis completo de todas las características que restan estudiar de la herramienta. Se proponen algunas posibles extensiones:

- Análisis de la integración y soporte de *RTC* con *ClearQuest*, *ClearCase* y *Subversion.*
- Análisis de la integración de *Rational Team Concert* con *Rational Quality Manager* y *Rational Requirements Composer*
- *Análisis de la versión cliente de Visual Estudio*
- *Análisis de los templates de los otros procesos incluidos en el software (*"Eclipse Way", " Cloudburst", " proceso Simple "*)*
- *Análisis de seguridad RTC en cliente (Web 2.0, Eclipse y Visual Estudio) y en el servidor*

Bibliografía

URL

[1] *Desarrollo iterativo y creciente* - http://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

[2] *Desarrollo rápido de aplicaciones (RAD)* - http://es.wikipedia.org/wiki/RAD

[3] *Fases del desarrollo de software* - http://es.wikipedia.org/wiki/Fases_del_desarrollo_de_software

[4] *Ingeniería de software* - http://es.wikipedia.org/wiki/Ingeniería_del_software

[5] *Desarrollo en cascada* - http://es.wikipedia.org/wiki/Modelo_en_cascada

[6] *Desarrollo en espiral* - http://es.wikipedia.org/wiki/Desarrollo_en_espiral

[7] *Modelo de prototipos* - http://es.wikipedia.org/wiki/Modelo_de_prototipos

[8] *Desarrollo por etapas* - http://es.wikipedia.org/wiki/Desarrollo_por_etapas

[9] *Desarrollo iterativo y creciente* - http://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

[10] *Problemas no resueltos de la ingeniería de software* - http://es.wikipedia.org/wiki/Problemas_no_resueltos_de_la_ingenier%C3%ADa_de_software

[11] *Ingeniería de software* - http://aprendeenlinea.udea.edu.co/lms/moodle/mod/resource/view.php?id=14273

[12] *Software* - http://es.wikipedia.org/wiki/Software#Proceso_de_creaci.C3.B3n_de_software

[13] *Agile Development* -

http://www.agilealliance.com/
[14] *Agile data Techniques* - http://www.agiledata.org/
[15] *Agile Methodology* - http://agilemethodology.org/
[16] *Best Practices for Software Development* - http://www.ambysoft.com/
[17] *Aplicando Scrum* - http://www.aplicandoscrum.com/
[18] *Desarrollo ágil de software* - http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software
[19] *Introduction to Scrum - An Agile Process* - http://www.mountaingoatsoftware.com/topics/scrum
[20] *Metodologías Ágiles (Scrum y OpenUp)* - http://formacion.morfeo-project.org/wiki/index.php/PT3:GPSL:UF6
[21] *Modelos de ciclo de vida en desarrollo de software* - http://www.acis.org.co/index.php?id=551
[22] *OpenUp* - http://en.wikipedia.org/wiki/OpenUP
[23] *Practical Agility* - http://practicalagility.blogspot.com/2008/11/waterfall-vs-rad-vs-rup-vs-agile-scrum.html
[24] *Proceso Unificado* - http://es.wikipedia.org/wiki/Proceso_Unificado
[25] *Scrum, la manera ágil de trabajar* - http://www.proyectosagiles.org/
[26] *Scrum Methodology* - http://scrummethodology.com/
[27] *Scrum* - http://es.wikipedia.org/wiki/Scrum
[28] *Ingeniería de software* - http://www.monografias.com/trabajos5/inso/inso.shtml
[29] Software configuration Management (SCM) - http://es.wikipedia.org/wiki/Software_configuration_management
[30] *Team Foundation Server Vs. Visual SourceSafe* - http://tfs2005.blogspot.com/2007/03/team-foundation-server-vs-visual-source.html
[31] *Xplanner* - http://www.xplanner.org/
[32] *Bazaar* - http://bazaar.canonical.com/en/
[33] *Sistema de control de versiones - GNU Bazaar* - http://picandocodigo.net/2008/sistema-de-control-de-versiones-gnu-bazaar/
[34] *Control de versiones distribuido (Un enfoque practico con Bazaar)* - http://www.slideshare.net/thegeekinside/control-de-versiones-distribuido
[35] *Formas de trabajo con Bazaar* - http://www.chuidiang.com/chuwiki/index.php?title=Formas_de_trabajo_con_Bazaar
[36] *Plastic SCM - Un genial sistema de control de versiones* - http://tecnobetas.com/plastic-scm-genial-sistema-de-control-de-versiones/
[37] *Git* - http://es.wikipedia.org/wiki/Git
[38] *Git (software)* - http://en.wikipedia.org/wiki/Git_%28software%29
[39] *Mercurial* - http://es.wikipedia.org/wiki/Mercurial
[40] *Mercurial (WhatsNew)* - http://mercurial.selenic.com/wiki/WhatsNew
[41] *Control distribuido de revisiones con Mercurial* - http://artigoo.com/control-distribuido-de-revisiones-con-mercurial
[42] *Mercurial: Software de control de versiones* - http://sentidoweb.com/2007/11/08/mercurial-software-de-control-de-versiones.php
[43] *Rational ClearCase* - http://www.rational.com.ar/herramientas/clearcase.html

[44] *IBM Rational ClearCase* - http://en.wikipedia.org/wiki/IBM_Rational_ClearCase
[45] *Team Foundation Server* - http://en.wikipedia.org/wiki/Team_Foundation_Server
[46] *Información general sobre Visual SourceSafe* - http://office.microsoft.com/es-es/access/HA102092363082.aspx
[47] *CVS* - http://es.wikipedia.org/wiki/CVS
[48] *Concurrent Versions System* - http://en.wikipedia.org/wiki/Concurrent_Versions_System
[49] *CVS sobre Windows* - http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=CVSsobreWindows
[50] *Visual SourceSafe 6.0.* - http://msdn.microsoft.com/es-es/library/cc434922%28VS.71%29.aspx
[51] *SourceSafe: Destructor de Código?* - http://mtycoders.com/sourcesafe-destructor-de-codigo/
[52] *Microsoft Visual SourceSafe* - http://es.wikipedia.org/wiki/Microsoft_Visual_SourceSafe
[53] *Visual SourceSafe: Microsoft's Source Destruction System* - http://www.highprogrammer.com/alan/windev/sourcesafe.html
[54] *Darcs* - http://en.wikipedia.org/wiki/Darcs
[55] *Una introducción a Darcs* - http://www.gulic.org/proyectos/darcs/darcs.html
[56] *Comparison of revision control software* - http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
[57] *Control de Versiones - CVS en proyectos .NET* - http://www.slideshare.net/dbaccess/control-de-versiones-cvs-en-proyectos-net
[58] *Jazz vision* - https://jazz.net/about/about-jazz-vision.jsp
[59] *Jazz architecture* - https://jazz.net/about/about-jazz-architecture.jsp
[60] *Visión general de Jazz. Innovación por medio de la colaboración* - http://www-01.ibm.com/software/es/rational/jazz/
[61] *Rational Team Concert. Overview* - http://jazz.net/projects/rational-team-concert/
[62] *Rational Team Concert Features* - http://jazz.net/projects/rational-team-concert/features/
[63] *Learn More About Using Rational Team Concert* - http://jazz.net/projects/rational-team-concert/learnmore/
[64] *What's New in Rational Team Concert* - http://jazz.net/projects/rational-team-concert/whatsnew/
[65] *Rational Team Concert Integrations* - http://jazz.net/projects/rational-team-concert/integrations/
[66] *Assessing Your Needs - Rational Team Concert* - http://jazz.net/projects/rational-team-concert/assessing-your-needs/
[67] *Rational Team Concert for System z* - http://jazz.net/projects/rational-team-concert-z/
[68] *Rational Team Concert for Power Systems Software* - http://jazz.net/projects/rational-team-concert-power/
[69] *Rational Change Management Express* - http://jazz.net/projects/rational-change-management-express/
[70] *Herramientas de colaboración informal para los equipos globales de desarrollo de software Informal* http://www.ibm.com/developerworks/ssa/rational/library/09/informalcollaborationtoolsforglobalsoftwaredevelopmentteams/?ca=drs-
[71 *How to use the Scrum project management method with IBM Rational Team Concert Version 1 and the Jazz platform* - http://www.ibm.com/developerworks/rational/library/08/0701_ellingsworth/
[72] *Rational Team Concert* - http://www-01.ibm.com/software/awdtools/rtc/

[73] *Instalando IBM Rational Team Concert Express-C* - http://www.ibm.com/developerworks/ssa/rational/library/08/0212_miller/index.html
[74] *Jazz and Rational Team Concert* - http://www.ibm.com/developerworks/offers/techbriefings/details/jrtc.html
[75] *Jazz in Concert—Jazz Platform and Rational Team Concert Make Sweet Music for Development Teams* - http://www.devx.com/ibm/Article/39497/0/page/1
[76] *Open Services for Lifecycle Collaboration* - http://open-services.net/html/Home.html
[77] *IBM Rational Team Concert 2.0.0.2* - http://publib.boulder.ibm.com/infocenter/rtc/v2r0m0/index.jsp?S_TACT=105AGY80&S_CMP=content
[78] *Rational Team Concert developer works*- http://www.ibm.com/developerworks/rational/products/rtc/
[79] *Rational Team Concert. Visual Studio Client (Video)* - https://jazz.net/pub/learn/videos/data/visual-studio-intro/rtcvs-beta.html
[80] *Biblioteca de Jazz* - http://www-01.ibm.com/software/es/rational/jazz/library.html
[81] *Trial: IBM Rational Team Concert Standard Edition* - http://www.ibm.com/developerworks/downloads/r/rtc/
[82] *Herramientas Case* - http://www.monografias.com/trabajos14/herramicase/herramicase.shtml
[83] *IBM Rational Team Concert and Jazz* - https://jazz.net/
[84] *Ayuda RTC* - *http://publib.boulder.ibm.com/infocenter/rtc/v2r0m0/index.jsp?topic=/com.ibm.team.install.doc/topics/c_ha.html*
[85] *IBM Rational Team Concert and Jazz* - http://www.ibm.com/

Papers

[86] *Apuntes teóricos de la materia "Ingeniería de Software I"* Facultad de Informática. UNLP
[87] *Apuntes teóricos de la materia "Ingeniería de Software II"* Facultad de Informática. UNLP
[88] *Apuntes teóricos de la materia "Arquitectura orientada a servicios"* Facultad de Informática. UNLP
[89] *Apuntes teóricos de la materia "Introducción a las bases de datos"* Facultad de Informática. UNLP
[90] *Metodología del análisis estructurado de sistemas* - Jesús Barranco de Areba
[91] *METODOLOGIAS DE DESARROLLO DE SOFTWARE* - Yenny Figueroa Medina - Universidad Manuela Beltrán
[92] *Metodologías Ágiles en el Desarrollo de Software* - José H. Canós, Patricio Letelier y Mª Carmen Penadés - Universidad Politécnica de Valencia
[93] *Procesos Ágiles* - Facultad de ingeniería (Universidad de la Republica - Uruguay)
[94] *Metodologías Tradicionales vs. Metodologías Ágiles* - Camilo Javier Solis Álvarez – Roberth Gustavo Figueroa Díaz (Universidad Técnica Particular de Loja)
[95] *Metodologías Tradicionales vs. Metodologías Ágiles* - Roberth G. Figueroa, Camilo J. Solís, Armando A. Cabrera (Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación)
[96] *Metodologías Ágiles* - Amaro Calderón, Sarah Dámaris. Valverde Rebaza. Jorge Carlos. (Facultad de Ciencias Físicas y Matemáticas. Universidad Nacional de Trujillo)
[97] *Flexibilidad con Scrum - Principios de diseño e implantación de campos de Scrum (Adaptando los procesos a la Empresa)* - Juan Palacio

[98] *El modelo Scrum* -
Juan Palacio (2006)
[99] SCRUM and XP from the Trenches - How we do Scrum
Henrik Kniberg (2007)
[100] *THE SCRUM PRIMER* -
Pete Deemer, Gabrielle Benefield, Craig Larman, Bas Vodde (2009)
[101] *Scrum Manager. Gestion de proyectos* -
Juan Palacio, Claudia Ruata (2009)
[102] *The Open Unified Process (OpenUP): Agile, Open Source, and Straightforward* -
Scott W. Ambler - Practice Leader Agile Development, IBM (2006)
[103] *Sistema de control de versiones descentralizados* -
Damián Noseda
[104] *Bazaar: Nueva generacion de sistemas de versionamiento distribuido* -
Martin Albisetti
[105] *Plastic SCM 2.0. Sistema distribuido* -
códicesoftware
[106] *Version Control with Subversion. For Subversion 1.6* -
Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato (2009)
[107] *Introducción a Rational ClearCase LT* -
Rosa María Carretero López (Universidad de Castilla-La Mancha)
[108] *Mondrian: Code Review on the Web* -
Guido van Rossum (2006)
[109] *El sistema de control de versions Mercurial* -
German Poo Camaño (2007)
[110] *Collaborative Software Development with Jazz and IBM Rational Team Concert* -
IBM Rational Technology Events
[111] *Ciclo de Vida de Desarrollo Colaborativo: Rational Team Concert, Requirement Composer y Quality Manager* -
Wendy Vazquez Acosta (Rational IT Specialist SSA) - IBM Forum 2009
[112] ***Jazz.** Nueva plataforma de colaboración para el desarrollo de software* -
IBM Software
[113] *IBM Rational Team Concert (Mejore la colaboración de sus equipos para un suministro de software automatizado, transparente y predecible)*
IBM Software
[114] *Innovación en el Desarrollo de Software a través de la Colaboración* -
Sergio Cedillo (Especialista Técnico en IBM Rational) - IBM Forum 2009
[115] *IBM Rational Team Concert for I* -
Armando Castillo (IT/Specialist SSA) - IBM Forum 2009
[116] *Jazz. El soporte definitivo para el soporte de factorías de software.*
Luis Reyes (Arquitecto de soluciones IBM)
[117] *IBM midmarket software. Buying and Selling guide* -
IBM Corp (2009)
[118] *The Jazz Tutorial - part 1, 2, 3, 4, 5, 6, 7, 8, 9, 10*
Akmal B. Chaudhri (Senior IT Specialist) - IBM Developer Skills Team